TAMPEREEN TEKNILLINEN YLIOPISTO

TUOMAS HAAPALA

TARTTUMISELEESEEN PERUSTUVAN LIIKEOHJAUKSEN TOTEUTUS MICROSOFT KINECTILLÄ

Diplomityö

Tarkastaja: Prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 9.1.2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HAAPALA, TUOMAS: Tarttumiseleeseen perustuvan liikeohjauksen toteutus Microsoft Kinectillä
Diplomityö, 42 sivua
Kesäkuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen
Avainsanat: Kinect, liikeohjaus, eletunnistus, OpenNI, NiTE, OpenCV

Monien graafisten sovellusten käyttö perustuu objektien siirtelyyn lokerosta toiseen. Perinteisesti tietokoneella niitä käytetään usein hiirellä ja niin kutsutulla drag-and-drop -ohjauksella. Mikäli tällaista sovellusta halutaan ohjata luonnollisemmalla tavalla, esimerkiksi julkisissa tiloissa, soveltuu kosketusnäyttö tähän tarkoitukseen hyvin.

Suuret kosketusnäytöt ovat kuitenkin edelleen todella kalliita. On myös mahdollista, että näyttö halutaan sijoittaa sellaiseen paikkaan, kuten lasin taakse, jonne fyysinen koskettaminen ei ole mahdollista. Nämä ongelmat ja muitakin voitaisiin ratkaista Microsoft Kinectiin perustuvalla liikeohjauksella.

Ongelmaksi Kinectin kanssa muodostuu, ettei sitä käyttäessä ole olemassa luonnollista tapaa valita objekteja. Ihminen siirtää kappaleita tarttumalla niistä kiinni, joten luonnollisena tapana valita objekti voitaisiin pitää tarttumiselettä. Sen tunnistaminen ei kuitenkaan onnistu Kinectin perustoiminnallisuudella.

Tässä diplomityössä on selvitetty, miten tarttumiseleen tunnistus onnistuu Kinectillä, ja arvioitu siihen perustuvaa liikeohjausta käytettävyyden kannalta. Teknisenä kontribuutiona on toteutettu ohjelmistokirjasto, joka mahdollistaa tarttumiseleeseen perustuvan liikeohjauksen. Lisäksi on toteutettu kaksi tätä ohjelmistokirjastoa käyttävää sovellusta sen arvioinnin avuksi.

Osoittautui, että tyydyttävän hyvin toimivan tarttumiseleen tunnistuksen rakentaminen Kinectillä onnistui suhteellisen yksinkertaisia tietokonenäkömenetelmiä käyttäen. Lisäksi saatiin selville, että tarttumiseleeseen perustuva liikeohjaus oli luonnollinen ja intuitiivinen käyttää. Suurin ongelma liikeohjauksessa oli käsien todella nopea väsyminen. Kun tulevaisuudessa tällainen ohjaus saadaan toimintavarmaksi, on se varteenotettava ohjaustapa sovelluksille, joita ei tarvitse käyttää pitkää aikaa kerrallaan.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HAAPALA, TUOMAS: Implementing grab gesture based motion control using Microsoft Kinect
Master of Science Thesis, 42 pages
June 2013
Major: Software engineering
Examiner: Professor Tommi Mikkonen
Keywords: Kinect, motion control, gesture recognition, OpenNI, NiTE, OpenCV

There are many graphical applications whose user interface is based on moving objects from one location to another. Traditionally with PCs this kind of software is often controlled with mouse and so called drag-and-drop controlling. When there is need for more natural way of controlling, for example in public places, drag-and-drop can be accomplished naturally using a touchscreen.

However, larger touchscreens are still very expensive. Sometimes it is also preferred to have the screen in an isolated place, for example behind a glass, where it cannot be physically touched. These problems and more can be addressed using Microsoft Kinect based motion controlled user interface.

With Kinect problem arises from the fact that there is no natural way to select objects. Humans move objects around by grabbing them, so grab gesture could be thought as potential way to naturally select objects. However, Kinect's basic functionality does not provide means to detect grab gesture.

This thesis studies the possibility of detecting grab gesture with Kinect, and also the usability of motion controlled user interface which includes grab gesture object selection. As a technical contribution, an open source software library which provides such functionality is developed.

It turns out that implementing a satisfyingly working grab gesture detection can be done using fairly simple computer vision methods. It was also learned that motion controlled user interface with grab gesture object selection is indeed natural and intuitive to use. Biggest problem with this kind of controlling was the fact that the user's arms worn out very quickly. When technology advances and motion controlling with flawless grab detection is achieved, it is very competitive alternative to touchscreens, as long as the software controlled is not meant to be used for long time straight.

ALKUSANAT

Suuret kiitokset Nemein Oy:lle, ja etenkin Henri Bergiukselle, jonka ideat, kannustus ja konkreettinen apu tekivät tämän diplomityön toteuttamisen mahdolliseksi.

Kiitokset myös kaikin puolin loistavalle tarkastajalle, professori Tommi Mikkoselle.

Ja luonnollisesti iso kiitos myös kotiväelle.

SISÄLLYS

1 JOHDANTO

Monien graafisten sovellusten käyttö perustuu objektien siirtelyyn lokerosta toiseen. Tämä siirtely toteutetaan usein hiirellä tai kosketusnäytöllä niin, että ensin siirrettävä objekti valitaan painamalla hiiren nappi pohjaan sen kohdalla tai painamalla sormi objektin kohdalle, ja objektin ollessa valittuna sitä voidaan siirrellä ruudulla liikuttamalla hiirtä tai sormea. Kun hiiren nappi vapautetaan tai sormi otetaan pois kosketusnäytöltä objektin siirto päättyy siihen kohtaan (engl. drag-and-drop).

Kosketusnäytöllä tällaisen raahaussiirtelyn käyttäminen on luonnollista. Siihen liittyy kuitenkin myös muutamia rajoituksia. Vaikka hinnat ovatkin tulleet viime aikoina alaspäin, suuret kosketusnäytöt ovat edelleen todella kalliita. Hinnan lisäksi näytön kokoa rajoittaa ihmisen fyysinen ulottuvuus. Etenkin julkisilla paikoilla kosketusnäyttöihin saattaa liittyä myös hygieenisiä ongelmia, kun käyttäjät koskettavat sitä fyysisesti. On myös mahdollista, että käyttäjät halutaan jostain syystä erottaa itse näytöstä, esimerkiksi niin, että näyttö sijaitsee liikkeen sisällä, ja sen käyttö tapahtuu liikkeen ulkopuolelta lasin toiselta puolelta.

Näihin ongelmiin ratkaisuna voidaan käyttää liikeohjausta. Microsoftin Kinect-liikeohjausjärjestelmää käyttämällä ei ohjauksessa tarvita apuna erillistä ohjainta. Sen hinta on noin kaksi sataa euroa, joten sen käyttö on paljon halvempaa verrattuna suureen kosketusnäyttöön. Lisäksi Kinectiä käytettäessä ei näyttöön tarvitse koskea, jolloin hygieniaongelmat poistuvat, ja sen käyttö onnistuu myös lasin läpi. Käyttäjän ulottuvuus ei myöskään rajoita mitenkään käytettävän näytön kokoa.

Kinectin käyttämisessä raahausohjauksessa ongelmaksi nousee se, että sitä käytettäessä ei ole luonnollista tapaa valita objekteja. Monissa Kinectin peleissä valinta tapahtuu niin, että halutun objektin päällä täytyy pitää kättä tietty aika, jotta objekti valitaan. Eräs käytetty tapa on myös tehdä valinta tekemällä kädellä painoliike eteenpäin. Näitä tapoja ei voida kuitenkaan pitää erityisen luonnollisina tapoina ihmiselle.

Ihminen siirtää esineitä tarttumalla niistä kiinni, joten luonnollisena valintaliikkeeksi voitaisiin kuvitella tarttumisliikkeen tekeminen kädellä, eli sulkemalla sormet yhteen. Tämän havainnointi ei kuitenkaan onnistu Kinectillä suoraan, vaan sen tarjoama tarkkuus rajoittuu raajojen seuraamisen tasolle. Kinectiltä saadaan kuitenkin myös syvyyskartta sen havainnoimasta maailmasta, jota analysoimalla on mahdollista selvittää, onko käyttäjän käsi tarttuneena.

Tässä diplomityössä on selvitetty, miten helposti ja hyvin tarttumiseleen tunnistus onnistuu Kinectillä. Lisäksi selvityksen alla on, miten hyvin tällainen ohjaustapa toimii

käytännössä. Diplomityön teknisenä kontribuutiona on toteutettu avoimen lähdekoodin ohjelmistokirjasto, joka mahdollistaa käden seurannan ja tarttumiseleen tunnistuksen. Arvioinnin avuksi on lisäksi toteutettu kaksi tätä ohjelmistokirjastoa käyttävää sovellusta.

Luvussa kaksi käydään läpi liikeohjauksen perusteet, siinä käytetyt tekniikat ja historia. Luvussa kolme esitetään Kinectin yleiskuvaus, laitteisto, käyttö, toimintaperiaate ja sovelluskehitys. Luku neljä käsittelee toteutetun ohjelmistokirjaston apuna käytettyjä valmiita kirjastokomponentteja. Luvussa viisi on esitelty yksityiskohtaisesti itse toteutettu ohjelmistokirjasto. Luvusta kuusi löytyvät arviointi ja jatkokehitysajatukset. Luvussa seitsemän esitetään yhteenveto.

2 JOHDATUS LIIKEOHJAUKSEEN

Tässä luvussa kuvataan yleisellä tasolla mitä liikeohjauksella tarkoitetaan, sekä liikeohjaukseen käytettyjä tekniikoita.

2.1 Perusteet

Liikeohjauksella ei ole tarkkaa vakiintunutta määritelmää, mutta yleisesti sillä tarkoitetaan järjestelmän kontrollointia käyttäjän liikkeiden perusteella. Perinteisesti liikeohjaus toteutetaan erilaisilla käyttäjän liikuttelemilla ohjaimilla, jotka välittävät käyttäjän liikkeet edelleen järjestelmälle. Tällöin jotta voidaan puhua liikeohjauksesta, täytyy ohjaamisen pääsääntöisesti olla mahdollista kaikissa kolmessa ulottuvuudessa. Näin esimerkiksi pelkästään pöydän tason suunnassa liikuteltavaa hiirtä ei lasketa liikeohjaimeksi. Toinen erityisesti viime aikoina yleistynyt tapa toteuttaa liikeohjaus on toteutus kokonaan ilman ohjainta, jolloin käyttäjä ohjaa järjestelmää esimerkiksi kättänsä liikuttelemalla.

Etenkin kuluttajille suunnatut liikeohjausjärjestelmät ovat perinteisesti liittyneet lähes yksinomaan videopeleihin. Viime aikoina, etenkin ohjaimettoman Kinectin ilmestymisen jälkeen, liikeohjausta on alettu soveltamaan monille muillekin aloille, kuten terveydenhuoltoon, robotiikkaan, kuvankäsittelyyn, lisättyyn todellisuuteen ja kolmiulotteiseen mallinnukseen. [1]

2.2 Tekniikat

Liikeohjattavissa järjestelmissä käyttäjän liikkeen mittaamiseen käytetään monia eri tekniikoita. Ohjaimellisissa järjestelmissä ohjaimet sisältävät erilaisia sensoreita, joilla niiden orientaatiota ja liikettä pyritään määrittämään. Yleisimmin käytettyjä sensoreita ovat kiihtyvyysmittarit, gyroskoopit ja magnetometrit. Liikeohjaus voidaan toteuttaa myös ilman ohjainta kuvaamalla käyttäjää kameroilla ja käyttämällä tietokonenäköön liittyviä tekniikoita käyttäjän liikkeiden analysointiin.

2.2.1 Sensorit

Kiihtyvyysmittareilla voidaan mitata kiihtyvyyttä eri suuntiin, joten sillä havaitaan ohjaimen liikuttaminen. Koska maapallon painovoima aiheuttaa vakiokiihtyvyyden sitä itseään kohti, voidaan kiihtyvyysmittarilla lisäksi määrittää sen hetkinen orientaatio maapallon pintaan nähden tutkimalla mihin suuntaan kiihtyvyys kohdistuu. Edelleen tästä seuraa, että kulmaa pyörittäessä maapallon pinnan suuntaisesti ei pystytä määrittämään kiihtyvyysanturilla, koska näillä eri orientaatioilla painovoiman aiheuttama kiihtyvyys osoittaa samaan suuntaan mittariin nähden. [2]

Perinteinen mekaaninen gyroskooppi koostuu tangosta ja siinä keskikohdastaan kiinni olevasta levystä, jotka on ripustettu kardaanisen rengassysteemin sisälle. Rengassysteemi pitää tankoa ja levyä paikallaan, mutta mahdollistaa niiden vapaan pyörimisen kaikkien kolmen akselin ympäri. Kun levy laitetaan pyörimään tangon ympäri, se pyrkii liikemäärämomentin säilymislakiin perustuen säilyttämään orientaationsa. Näin tanko ja levy pysyvät samassa orientaatiossa vaikka systeemiä ulkoisesti käännellään, ja kulmanopeus käänneltäessä voidaan määrittää. Perinteisen pyörivään levyyn perustuvan gyroskoopin lisäksi on kehitetty monia eri tyyppisiä eri ilmiöihin toimintansa pohjaavia gyroskooppeja. Nykyisin kuluttajille suunnatuissa elektronisissa laitteissa käytetään yleisimmin hyvin pieniä MEMS-teknologialla (engl. Micro Electro Mechanical Systems, suom. mikrosysteemit) valmistettuja värähtelyyn ja Corialis-ilmiöön perustuvia gyroskooppeja niiden hyvän saatavuuden, pienen koon ja halpuuden ansiosta. [3; 4]

Magnetometrilla voidaan mitata magneettikentän suuntaa ja voimakkuutta. Maapallon magneettikenttää hyväksikäyttäen niillä saadaan määritettyä laitteen orientaatio maapallon magneettiseen pohjoisnapaan nähden. Käytännössä orientaation määrittämisessä on ongelmia etenkin sisätiloissa niissä esiintyvien magneettisten häiriöiden johdosta. [5]

Erilaisilla sensoreilla on erilaiset vahvuudet ja heikkoudet. Näin ollen mikäli orientaatiota ja liikettä halutaan määrittää kokonaisvaltaisesti, niitä ei usein käytetäkään yksittäin, vaan useamman sensorin kombinaationa, jossa eri tyyppiset sensorit täydentävät toisiaan. Yksi yleisesti käytetty yhdistelmä on kiihtyvyysmittarin ja gyroskoopin kombinaatio. Karkeasti voidaan sanoa, että kiihtyvyysmittarit näyttävät pidemmän päälle oikein, mutta lyhyellä välillä häiriöalttiita, kun taas gyroskoopit ovat lyhyellä välillä tarkkoja, mutta pidemmällä välillä alkavat harhaantua. Tällöin paras tulos saavutetaan näiden kahden laitteen yhdistelmällä, jossa niitä voidaan käyttää toistensa kalibroimiseen. [6]

Käden ja sormien liikkeiden määrittämiseen voidaan käyttää käteen puettavaa datahansikasta (engl. data glove). Ne sisältävät erilaisia magnetismiin tai sähkömekanismiin perustuvia sensoreita, jotka seuraavat sormien ja niiden nivelien liikkeitä ja kulmia. Datahansikkaan avulla käden ja yksittäisten sormien asentojen määrittäminen onnistuu tarkasti. Niiden käyttöön liittyy kuitenkin myös monia ongelmia. Käyttäminen vaatii hansikkaan päälle pukemisen, joka vähentää ratkaisun luonnollisuutta ja käytettävyyttä. Lisäksi se aiheuttaa hygieenisiä ongelmia mikäli järjestelmää on tarkoitus käyttää julkisissa tiloissa. Datahansikkaat ovat myös kalliita valmistaa ja vaativat monimutkaisen kalibroinnin toimiakseen tarkasti. [7]

2.2.2 Kamerat

Käyttäjän liikkeitä voidaan mitata myös kuvaamalla tätä erilaisilla kameroilla. Tällöin kyseessä on tietokonenäkö-ongelma, jossa kameroiden kuvaamasta kuvasta ensin tunnistetaan ja erotetaan niiden osien liikkeet, joilla ohjausta on tarkoitus suorittaa, ja tämän jälkeen määritetään niiden liikkeitä. Mikäli käytössä on ohjain, voidaan siinä mahdollisesti olevien sensoreiden lisäksi sen sijaintia kameraan nähden määrittää lisäämällä siihen joku helposti tunnistettava osa, esimerkiksi tietynvärinen pallo ohjaimen päähän, joka paikallistetaan kuvasta kuvankäsittelyn keinoin.

Kameroihin perustuva liikeohjausjärjestelmä voidaan toteuttaa myös ilman ohjainta. Tällöin tunnistus ei monestikaan ole triviaali tehtävä, ja se hankaloituu sitä myötä, mitä pienempiä yksityiskohtia halutaan käyttää ohjaamiseen. Liikeohjaus on mahdollista toteuttaa jo yksittäisellä tavallisella kaksiulotteista kuvaa tuottavalla RGB-kameralla. Tehtävä helpottuu kuitenkin huomattavasti, jos kuvaan saadaan myös syvyysinformaatiota esimerkiksi useammalla eri paikasta kuvaavalla kameralla, tai infrapunasäteisiin perustuvalla syvyyskameralla.

Pelkästään kameroihin perustuvat tekniikat voidaan jakaa kahteen päätyyppiin. Kolmiulotteisiin malleihin perustuvat tekniikat pyrkivät ensin mallintamaan kuvattavan kohteen kolmiulotteisena kappaleena, jonka eri osien liikkeitä analysoimalla voidaan sitten tunnistaa erilaisia eleitä. Toinen vaihtoehto on analysoida kuvaan liittyviä piirteitä, esimerkiksi siinä esiintyviä kuvioita. Tähän yhdistetään monesti koneoppimistekniikoita, joissa saatuja parametreja verrataan tunnettuihin malleihin. [8]

Tässä diplomityössä toteutettu ohjelmistokirjasto käyttää tarttumisotteen tunnistukseen kuvan analysointia, eikä rakenna kädestä kolmiulotteista mallia. Tähän ratkaisuun päädyttiin siksi, että käden tarttuneena olemisen analysointi onnistuu hyvin kaksiulotteisesta kuvasta, eikä kolmiulotteisen mallin tarjoamalla lisäinformaatiolla, esimerkiksi sormien tarkoilla sijainneilla, saavuteta lisähyötyä. Lisäksi tarttumisotteen tunnistuksen haluttiin toimivan mahdollisimman kaukaa. Vaarana kolmiulotteisen mallin käytön kanssa nähtiin, että Kinectin kameroiden rajallisen resoluution takia toimintaetäisyys jäisi pienemmäksi kuin kaksiulotteista kuvaa analysoimalla. Valittu toteutustekniikka on myös huomattavasti yksinkertaisempi toteuttaa kuin kolmiulotteinen malli. Toteutustekniikka on kuvattu yksityiskohtaisesti kohdassa 5.5. [8]

2.3 Käytännön toteutuksia

Varhaisimmat kuluttajille suunnatut liikeohjausjärjestelmät liittyivät yksinomaan videopeleihin. Näistä ensimmäisenä voidaan pitää Datasoft-nimisen yrityksen vuonna 1981

Atari 2600- ja Commodore 64 -tietokoneille julkaisemaa ohjauslaitetta nimeltä Le Stick, joka mahdollisti ohjauksen kahdeksaan suuntaan. Sille ei kuitenkaan ikinä julkaistu yhtään peliä. [9]

Seuraavat kolme liikeohjausjärjestelmää olivat Nintendon NES-pelikonsolilleen julkaisemia lisälaitteita. Vuonna 1986 se julkaisi PowerPad-nimeä kantavan maton, joka havainnoi käyttäjän sen päällä tekemiä askeleita ja hyppyjä. Seuraavaksi oli vuorossa vuonna 1989 julkaistu hansikas PowerGlove, joka oli ensimmäinen käyttäjän käden liikkeitä seuraava laite. Samana vuonna Nintendo julkaisi myös U-Force -nimisen lisälaitteen. Tämä oli ensimmäinen liikeohjausjärjestelmä, jossa käyttäjän liikkeitä seurattiin infrapunasäteiden avulla. [9]

Vuonna 1995 Sega julkaisi ensimmäisen liikeohjaukseen perustuvan järjestelmänsä, Sega Nemesis -pelikonsolille tarkoitetun lisälaitteen Sega Activator. Sitä voidaan pitää ensimmäisenä järjestelmänä, jossa käyttäjä ohjasi vapaasti ilman ohjainta. Toiminta perustui levyyn, jonka päällä käyttäjä seisoi. Levyn reunoilla oli infrapunasäteisiin perustuvat sensorit, jotka havaitsivat käyttäjän raajojen liikkeet. Sega julkaisi myös vuonna 1999 Dreamcast-pelikonsolilleen Samba de Amigo -nimisen pelin, jota ohjattiin marakassia muistuttavilla ohjaimilla. [9]

Sony julkaisi vuonna 2003 Playstation 2 -pelikonsolilleen Playstation EyeToy -nimellä kulkevan lisälaitteen, joka mahdollisti ohjaimettoman liikeohjauksen. Siinä liikkeiden mittaaminen perustui tavalliseen RGB-kameraan. Yksikään näistä varhaisista liikeohjausjärjestelmistä ei saavuttanut suurta suosiota, ja monia niistä pidettiin todella huonoina. [9]

Ensimmäinen liikeohjaukseen perustuva laajalle levinnyt ja kaupallisesti menestynyt järjestelmä oli Nintendon vuonna 2006 julkaisema Wii -pelikonsoli. Sen liikeohjauksen tunnistus perustuu television lähelle sijoitettavaan Sensor Bariin ja Wii Remote -ohjaimeen. Sensor Bar sisältää viisi infrapunavaloa, jotka ohjain havaitsee valosensorillaan. Havaituista infrapunavalojen sijainneista voidaan määrittää ohjaimen suunta ja sijainti suhteessa Sensor Bariin, ja näin ohjainta voi käytännössä käyttää ruudulla olevien elementtien osoittamiseen. Kuvakennon lisäksi ohjain sisältää myös kolmiulotteisen kiihtyvyysmittarin. Vuonna 2009 Nintendo julkaisi Wii MotionPlus -lisälaitteen Wii Remote -ohjaimeen, joka paransi sen liikkeentunnistusta lisäämällä ohjaimeen gyroskoopin. [10]

Toinen suosittu liikeohjausjärjestelmä on Sonyn vuonna 2010 julkaisema PlayStation Move. Se koostuu PlayStation Move -liikeohjaimesta ja PlayStation Eye -kamerasta. Kamera sijoitetaan television tai näytön lähelle kuvaamaan pelialueelle päin ja kullakin pelaajalla on yksi tai kaksi liikeohjainta käsissään. Liikeohjain pitää sisällään kolmen

akselin kiihtyvyysmittarin, kolmen akselin gyroskoopin ja magnetometrin, joiden avulla sen liikkeet ja orientaatio saadaan määritettyä. Näiden lisäksi ohjaimen päässä on muutaman senttimetrin halkaisijaltaan oleva pallo, joka on valaistu sisältä ja joka voi vaihtaa väriään. Tämä pallo paikallistetaan PlayStation Eye -kameran kuvaamasta kuvasta, jolloin saadaan määritettyä ohjaimen sijainti pelialueella. Kamera on normaali RGB-kamera, joten paikallistaminen suoritetaan kaksiulotteisen konenäön avulla. Kameraa käytetään myös kuvien ja videokuvan ottamiseen pelaajasta. [11]

Nintendon Wiin ja Sonyn PlayStation Moven lisäksi kolmas suosittu liikeohjausjärjestelmä on Microsoftin julkaisema Kinect. Se eroaa kahdesta muusta ratkaisevasti siten, että siinä ei käytetä minkäänlaisia pelaajalla olevia ohjaimia. Tässä diplomityössä käytetään nimenomaan Kinectiä, joten sen tarkempi esittely löytyy seuraavasta luvusta.

3 KINECT

Tässä luvussa esitellään diplomityössä käytetyn liikeohjausjärjestelmän, Kinectin, yleiskuvaus, käyttö, laitteisto, toimintaperiaate ja sovelluskehitys.

3.1 Yleiskuvaus

Kinect on Microsoftin vuonna 2010 julkaisema liikeohjausjärjestelmä. Se eroaa aiemmista liikeohjausjärjestelmistä sillä, että siinä erillistä ohjainta ei tarvita, vaan ohjaaminen tapahtuu pelkästään kehoa ja sen osia liikuttelemalla. Kinectin liikkeentunnistusteknologian on kehittänyt israelilainen PrimeSense. [12]

Alun perin Kinect julkaistiin käytettäväksi Microsoftin Xbox 360 -pelikonsolin kanssa nimellä Kinect for Xbox 360. Vuonna 2011 Microsoft julkaisi laitteesta Windows-käyttöjärjestelmän kanssa käytettävän Kinect for Windows -version, joka on laitteistoltaan sama kuin edellinen versio, mutta siinä on eri firmware eli laiteohjelmisto. Käytännössä tämä tarkoittaa, että versioissa on pieniä eroja ominaisuuksissa, eikä niillä voida suoraan käyttää kaikkia samoja sovelluskehitystyökaluja. Tässä tutkimuksessa käytetään vanhempaa versiota Kinect for Xbox 360, ja tekstissä puhuttaessa Kinectistä tarkoitetaan nimenomaan tätä vanhempaa versiota ellei toisin mainita. Kuvassa 1 on esitetty kyseinen laite. [13]

Kuva 1: Kinect for Xbox 360.[14]

Kinect on ollut suuri kaupallinen menestys. Ensimmäisen 60 päivän aikana sitä myytiin yli kahdeksan miljoonaa kappaletta, tehden siitä Guinnessin ennätysten kirjan nopeimmin myyneen kuluttajaelektroniikkalaitteen. Maaliskuussa 2011 Kinectejä oli myyty jo yli 10 miljoonaa kappaletta. Kinect sisältää hintaansa nähden paljon hienostunutta tekniikkaa, ja se onkin saavuttanut paljon suosiota videopelaajien lisäksi myös erilaisten hakkereiden ja virittelijöiden keskuudessa. [15]

3.2 Käyttö

Kinect tulee sijoittaa television tai näytön läheisyyteen 0,6 - 1,8 metrin korkeuteen lattiasta niin, että se pystyy kääntymään pystysuunnassa vapaasti. Pelaajien pitää olla vähintään metrin pituisia, ja pelialueella tulisi olla kokoa laitteesta poispäin 1,8 metriä yhdellä pelaajalla ja 2,4 metriä kahdella pelaajalla. Kinectillä tulisi olla esteetön näkyvyys pelialueelle, eikä välissä saisi olla mitään läpinäkyvääkään materiaalia, kuten lasia. Kinectin oikeaoppinen sijoittaminen on esitettynä kuvassa 2. [16]

Kinectiä käytettäessä pitäisi valaistuksen olla hyvä ja tasainen huoneessa jossa sitä käytetään. Suoraa auringonvaloa ei saisi tulla pelialueelle. Suosituksena on myös, ettei pelaaja käyttäisi löysiä vaatteita, ja joidenkin mustien vaatteiden tiedetään aiheuttavan ongelmia. [17]

Käytännössä havaittiin, että Kinect voi kuvata pelialuetta lasin läpi ilman ongelmia, ainakin silloin kun lasi on Kinectin välittömässä läheisyydessä. Havaittiin myös, että valaistus sai vaihdella normaalien sisäolosuhteiden puitteissa aiheuttamatta suurempia ongelmia Kinectin toimintaan, mutta suora auringonvalo sen sijaan sekoitti toiminnan hyvinkin helposti. Erilaisilla vaatteilla ei huomattu olevan vaikutusta, poislukien tarttumisohjaukseen liittyvä ongelma pitkien hihojen kanssa, jota on käsitelty kohdassa 6.2.

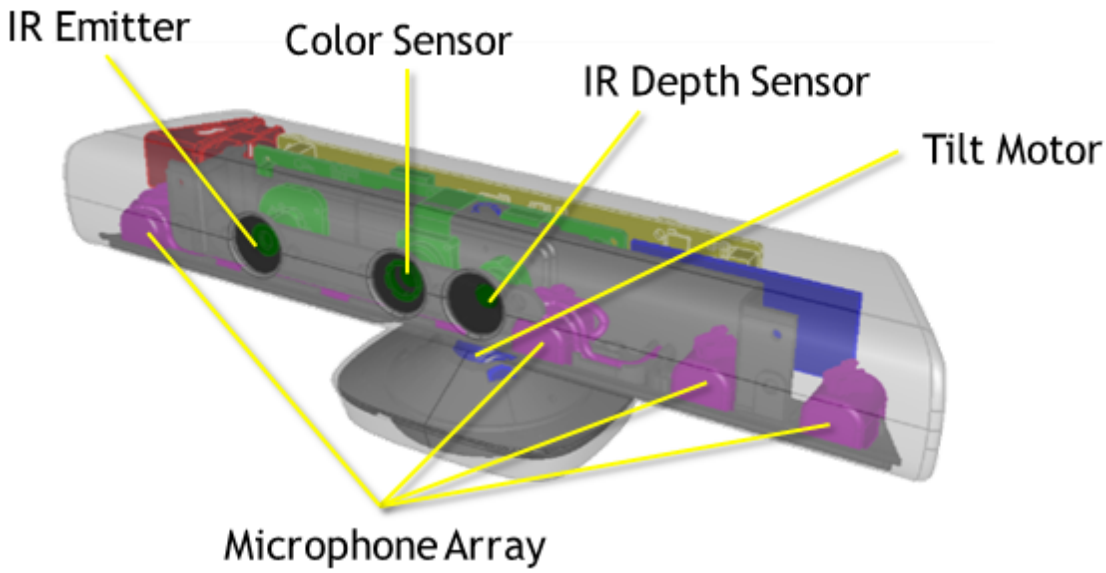
Kuva 2: Ohjeet Kinectin sijoittamiseen.[18]

3.3 Laitteisto

Tämä kohta perustuu lähteeseen [19]. Kinectin tärkeimmät osat ovat:

- **RGB-kamera** jossa on 1280x960 resoluutio. Se mahdollistaa normaalin videokuvauksen.
- **Infrapunalähetin (IR Emitter),** joka lähettää infrapunasäteitä pelialueelle.
- **Infrapunavastaanotin (IR Depth Sensor)**, mustavalkoinen CMOS-kenno, joka havaitsee infrapunalähettimen lähettämiä infrapunasäteitä kun ne heijastuvat takaisin laitteeseen. Tämä mahdollistaa syvyyskuvan muodostamisen.
- **4 mikrofonia (Microphone Array),** jotka toimivat yhteistyössä, ja mahdollistavat äänilähteen paikan ja saapuvien ääniaaltojen suunnan määrittämisen.
- **Kolmen akselin kiihtyvyysmittari,** jonka avulla Kinectin orientaatio saadaan määritettyä.

Kinectin laitteiston teknisiä yksityiskohtia on listattuna taulukossa 1. Kuva 3 esittää Kinectin eri osien sijainnit.



Kuva 3: Kinectin tärkeimmät osat [19]

Näkökenttä	43 astetta vaakatasossa, 57 astetta pystytasossa
Kallistuskulma vaakatasossa	+- 27 astetta
Kuvataajuus (sekä syvyys- että RGB-kamera)	30Hz
Audioformaatti	16-kHz, 24-bittinen monopulssimodulaatio (PCM)
Audio-sisääntulon ominaisuuksia	4 mikrofonia joissa 24-bittinen A/D-muunnin, kaiunpoisto, kohinanpoisto
Kiihtyvyysmittarin ominaisuuksia	2G/4G/8G kiihtyvyysmittari joka on konfiguroitu 2G:lle, tarkkuusyläraja 1 aste

Taulukko 1: Kinectin laitteiston teknisiä yksityiskohtia.

3.4 Toimintaperiaate

Käyttäjille Kinect näyttäytyy mahdollistaen monia kehittyneitä ominaisuuksia, kuten käyttäjien käsien ja kehojen seuraamisen. Itse Kinect-laite vastaa kuitenkin ainoastaan kolmiulotteisesta skannauksesta tuottaen kuvaamastaan alueesta syvyyskartan. Se on kuin 640x480-resoluution kuva, jossa pikselin arvo vastaa sen etäisyyttä sensorista. Korkeamman tason palvelut tuotetaan tätä syvyyskarttaa hyödyksi käyttäen vasta ohjelmallisesti Xboxissa tai tietokoneessa, johon Kinect on kytketty.

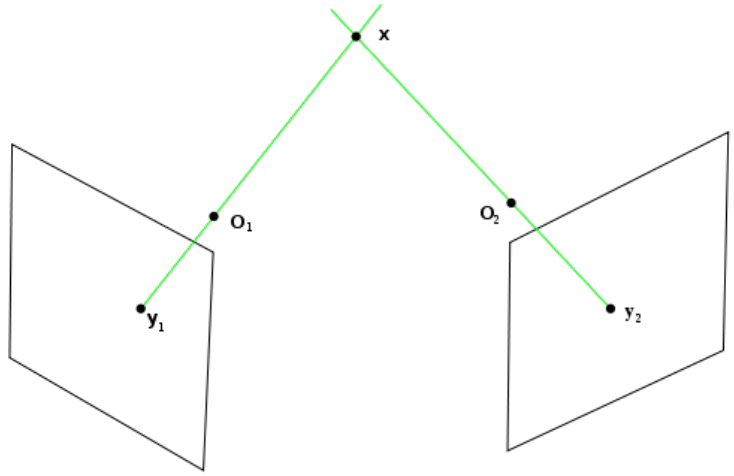
Syvyyskartta muodostetaan Kinectissä käyttämällä sen infrapunalähetintä ja infrapunavastaanotinta. Perinteisesti muissa vastaavissa järjestelmissä käytetty infrapunasäteillä toteutettu etäisyyden mittaus on perustunut matkan määrittämiseen mittaamalla aika, joka kestää, kun säteet matkaavat lähettimestä kohteeseen ja siitä edelleen heijastumalla takaisin vastaanottimeen (engl. Time Of Flight, TOF). Varsinkin Kinectin ilmestymisen aikoihin vuonna 2010 esiintyi yleinen harhaluulo, että myös Kinectin toiminta perustuu säteiden matkaaman ajan mittaukseen. Tämä ei kuitenkaan pidä paikkaansa, vaan Kinectissä käytetään PrimeSensen kehittämää, eräänlaista strukturoituun valoon perustuvaa tekniikkaa, josta PrimeSense itse käyttää nimitystä Light Coding. [20]

Kinectissä käytettävän tekniikan yksityiskohtia tai siinä käytettyjä algoritmeja ei ole julkistettu yleiseen tietoon. Sen toimintaperiaatteet on kuitenkin saatu selvitettyä pääpiirteittäin. Pohjimmiltaan käytetty tekniikka perustuu kolmiomittaukseen. [21]

Perinteisessä konenäköön liittyvässä kolmiomittauksessa käytetään vähintään kahta kameraa, joilla kuvataan samaa kohdepistettä eri paikoista. Jokaista kameran muodostamassa kuvassa olevaa pistettä vastaa kameran polttopisteen ja kuvattavan pisteen kautta kulkeva projektiosuora kolmiulotteisessa avaruudessa. Tällä suoralla olevat kolmiulotteiset pisteet kuvautuvat siis samaksi kaksiulotteiseksi pisteeksi kameran kuvassa. Näin

mikäli tiedetään, mikä piste kameran kuvassa vastaa mitattavaa kohdepistettä, voidaan kohdepisteen kolmiulotteinen sijainti määrittää yksinkertaisella algebralla selvittämällä sitä vastaavien projektiosuorien leikkauspiste.

Kolmiomittaus on esitetty graafisesti kuvassa 4. Kuvan alareunassa olevat nelikulmiot esittävät kahden kameran muodostamia kuvia. Piste **x** on kohdepiste, jonka kolmiulotteista sijaintia ollaan määrittämässä. Pisteet **o1** ja **o2** ovat kameroiden polttopisteitä, ja pisteet **y1** ja **y2** paikoissa, joihin kohdepiste kuvautuu kameroiden muodostamiin kuviin. Projektiosuorat on merkitty vihreällä.



Kuva 4: Kolmiomittaus.[21]

Kinectin käyttämässä tekniikassa toisena kuvana käytetään sen infrapunavastaanottimen muodostamaa kuvaa, joten tältä osin se on yhtenevä perinteisen kolmiomittauksen kanssa. Toista kuvaa ei sen sijaan kuvata mittausaikana, vaan toisena kuvana käytetään ennalta muodostettua vakiokuvaa. Infrapunasäde projektoi tämän vakiokuvan mitattavaan kohteeseen. Projektoitava kuva koostuu pseudosatunnaisesta pistekuviosta, jossa jokaisen pisteen ympäristössä olevat pisteet muodostavat sille pisteelle ominaisen uniikin rakenteen. Tällainen kuva soveltuu hyvin tilanteeseen, jossa kuvan tiettyä osaa analysoimalla halutaan selvittää, missä kohdassa se sijaitsee koko kuvassa. Näin ollen kun tietyn pisteen etäisyys vastaanottimen kuvassa halutaan määrittää, selvitetään ensin mitä kohtaa projektoidussa kuvassa se vastaa, jonka jälkeen pisteen etäisyys voidaan määrittää kolmiomittausta vastaavasti. Kun infrapunalähetin ja vastaanotin ovat samassa linjassa, riittää etäisyyden määrittämiseen, kun verrataan projektoidun kuvan ja vastaanottimen muodostaman kuvan välistä vaakasuuntaista siirtymää. [21]

Kuvassa 5 on esitettynä Kinectin lähettämä pseudosatunnainen infrapunakuvio kuvattuna infrapunakameralla.

Kuva 5: Kinectin lähettämä infrapunakuvio. [22]

3.5 Sovelluskehitys

Julkaisunsa jälkeen Kinectin sovelluskehitys oli suljettua, eikä sen toiminnasta ollut saatavilla tietoja. Hyvin pian Adafruit-niminen yritys julkaisikin Kinectin hakkerointikilpailun aluksi 2000 dollarin, sittemmin 3000 dollarin pääpalkinnolla. Tarkoituksena oli kehittää avoimen lähdekoodin ajuri Kinectille, eli käytännössä ohjelma, joka pystyi lukemaan Kinectin lähettämää RGB- ja infrapunakuvaa. Noin viikon päästä kilpailun alkamisesta sille ilmoitettiin löytyneen voittaja. Tämän pohjalta syntyi Open Kinect -yhteisö, joka kehittää avoimen lähdekoodin ohjelmistoja Kinectille. Yhteisön tärkein tuotos on libfreenect-kirjasto, joka mahdollistaa Kinectin käytön Windowsilla, Linuxilla ja OS X:llä. [23]

Alun perin Microsoftin kanta Adafruitin hakkerointikilpailuun oli jyrkän kielteinen. Se tiedotti rakentaneensa Kinectiin monia laitteisto- ja ohjelmistosuojauksia, joiden tarkoitus on estää sen peukalointi. Microsoft myös uhkasi mahdollisia hakkereita jopa oikeustoimilla. Kilpailun ratkettua Microsoft muutti kuitenkin kantaansa täydellisesti, ja ilmoitti jättäneensä Kinectin tarkoituksella avoimeksi niin, että sen lähettämää dataa

voisi lukea myös kolmannen osapuolen ajureilla. Microsoft tiedotti myös olevansa innoissaan hakkereiden kekseliäisyydestä, ja että ketään ei tultaisi haastamaan oikeuteen asian tiimoilta. Myöhemmin kuitenkin paljastui, että Microsoftin Kinect-kehitystiimin keskeinen jäsen oli alun perin lähestynyt Adafruitia kilpailuidean kanssa, ja maksoi sen 3000 dollarin pääpalkinnon. Microsoft julkaisi myös omat Kinect-kehitystyökalunsa Windows-käyttöjärjestelmälle, Kinect for Windows SDK:n, keväällä 2011. [24; 25]

Libfreenectin ja Kinect for Windows SDK:n lisäksi kolmas merkittävä kehitysympäristö, joka mahdollistaa sovelluskehityksen Kinectille, on avoimen lähdekoodin OpenNI. Sitä myös käytetään tähän työhön liittyen toteutetussa ohjelmistokirjastossa, joten sen tarkempi esittely löytyy kohdasta 4.2.

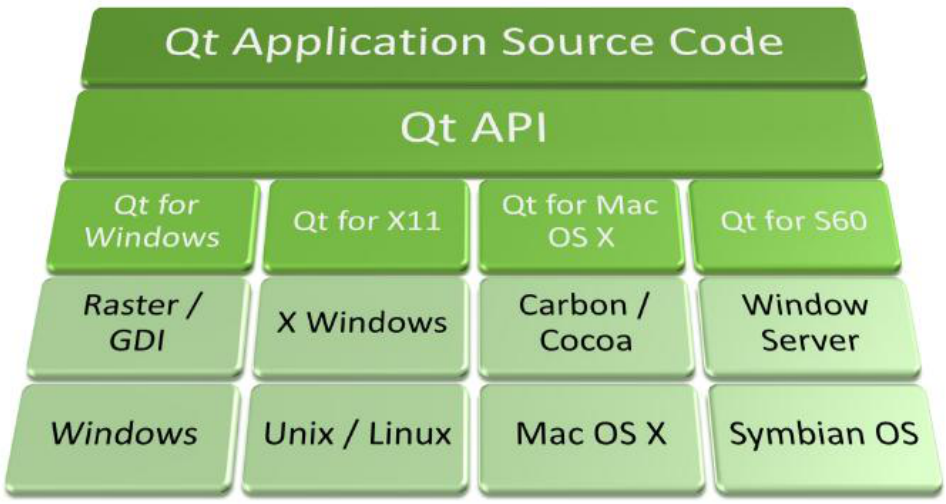
4 KÄYTETYT KIRJASTOKOMPONENTIT

Tässä luvussa kuvataan sovelluskehys sekä ohjelmistokirjastot, joita käytettiin diplomityön teknisenä kontribuutiona toteutetussa ohjelmistokirjastossa.

4.1 Qt

Qt on alun perin norjalaisen Trolltechin kehittämä laitteistoriippumaton avoimen lähdekoodin sovelluskehys, jonka ensimmäinen julkinen versio julkaistiin vuonna 1995. Nokia osti Qt:n vuonna 2008 ja kehitti sitä siitä lähtien etenkin mobiilipuolella voimakkaasti. Sitä käytettin Nokian Symbian- ja Maemo-mobiilikäyttöjärjestelmien sovelluskehitykseen. Kun vuonna 2012 Nokia oli hylännyt omien käyttöjärjestelmiensä Symbianin ja Maemon jatkajaksi kaavaillun Meegon kehityksen, ei Qt:lle ollut enää yhtiössä sijaa ja se myytiin Digialle, joka jatkaa sen kehitystä. Qt:n kehityksessä on lisäksi aktiivisesti mukana Qt Project -yhteisö. [26]

Qt:n ytimen muodostavat kattavat C++-pohjaiset ohjelmistokirjastot. Lisäksi Qt tarjoaa monia aputyökaluja sovelluskehittäjille, tärkeimpänä Qt Creator -nimellä kulkeva integroitu ohjelmointiympäristö (IDE), joka on kehitetty ja optimoitu nimenomaan Qt-ohjelmien kehitykseen. Qt-ohjelmat kirjoitetaan C++-kielellä, jota on laajennettu monilla ominaisuuksilla. Ohjelman lähdekoodi käännetään ensin Qt:n omalla metakääntäjällä (MOC) tavalliseksi C++-ohjelmakoodiksi, josta se sitten voidaan kääntää edelleen tavallisella C++-kääntäjällä suoritettavaksi ohjelmaksi. Qt:n käyttö muilla ohjelmointikielillä on mahdollista siihen kehitettyjen sidosrajapintojen avulla, joita löytyy kaikille yleisimmille kielille. [26]



Kuva 6: Qt:n korkean tason arkkitehtuuri [27]

Qt:n kantava ajatus on, että sama lähdekoodi on mahdollista kääntää eri alustoille tekemättä siihen muutoksia. Työpöytäpuolella tuettuja alustoja ovat Windows, Linux, Mac OS ja monet Unix-järjestelmät, jotka käyttävät X11-ikkunointijärjestelmää. Lisäksi mobiilipuolella ainakin osittainen tuki on olemassa Symbianille, Maemolle ja Meego OS:lle. Kuvassa 6 on kuvattu Qt:n korkean tason arkkitehtuuri. [26]

Yksi suosittu ominaisuus, jonka Qt lisää tavalliseen C++-kieleen, on sen mahdollistama signal/slot -mekanismi. Siinä olioilla on erilaisia signaaleita, joita ne lähettävät erilaisissa tilanteissa. Näitä erilaisia signaaleita on myös mahdollista määritellä lisää. Lisäksi olioilla on paikkoja, jotka vastaanottavat näitä signaaleja (engl. slot), ja näitä on niin ikään mahdollista määritellä lisää. Signaalin voi kytkeä yhteen tai useampaan paikkaan, ja Qt pitää huolen, että kun signaali lähetetään, välitetään se kaikille olioille, joihin kyseinen signaali on kytkettynä. Tämä ominaisuus tarjoaa joustavan ja helppokäyttöisen vaihtoehdon callback-funktioille, ja on erityisen hyödyllinen asynkronisessa järjestelmässä. [26]

Versiosta 4.7 alkaen Qt:ssa on ollut mukana Qt Quick -ympäristö, joka mahdollistaa ohjelmien toteuttamisen Javascriptiä muistuttavalla dynaamisesti tyypitetyllä QML-kielellä, joka on erityisesti kehitetty Qt:ta varten. Qt Quick eroaa muusta Qt:sta perustavanlaatuisesti siten, että QML-ohjelmakoodia ei C++-pohjaisten osien tavoin käännetä ensin binääriksi, vaan se tulkataan ajonaikaisesti virtuaalikoneella. QML mahdollistaa etenkin käyttöliittymien nopean kehityksen vähällä määrällä ohjelmakoodia. Ohjelmia on mahdollista kehittää pelkästään QML-kielellä, mutta useimmissa tapauksissa parhaaseen lopputulokseen päästään, kun QML-kielen lisäksi ohjelmassa toteutetaan suorituskykykriittisimmät osat perinteisellä Qt C++-kielellä. Jatkossa, etenkin versiosta 5.0 eteenpäin, Qt Quickin rooli Qt:ssa on kasvamassa. [28]

Vaikka Qt on alun perin tarkoitettu yksinomaan graafisten käyttöliittymien toteuttamiseen, on se kehityshistoriansa aikana hiljalleen muodostunut yhä monipuolisemmaksi kehitysympäristöksi myös muunlaisille ohjelmille. Nykymuodossaan Qt on hyödyllinen apuväline monenlaisten ohjelmien kehitykseen, jopa täysin ei-graafisten. Se sisältää kattavat laitteistoriippumattomat kirjastot muun muassa tiedostojen, verkkoyhteyksien ja ajastimien käyttöön.

Qt:lla tehdyillä ohjelmilla on mahdollista käyttää joko kaupallista tai avoimeen lähdekoodiin perustuvaa lisenssiä. Alun perin kaupallisen lisenssin lisäksi tarjolla oli ainoastaan LGPL-lisenssi, joka edellytti, että myös tehtyjen ohjelmien lähdekoodit on julkaistava. Sittemmin LGPL-lisenssin rinnalle on tullut mahdollisuus käyttää GPL-lisenssiä, jossa tehtyjen ohjelmien lähdekoodeja ei tarvitse julkaista, jos käyttää Qt:n kirjasto-

ja ajonaikaisella linkkauksella. Tämän diplomityön yhteydessä on käytetty Qt:n versiota 4.8. [29]

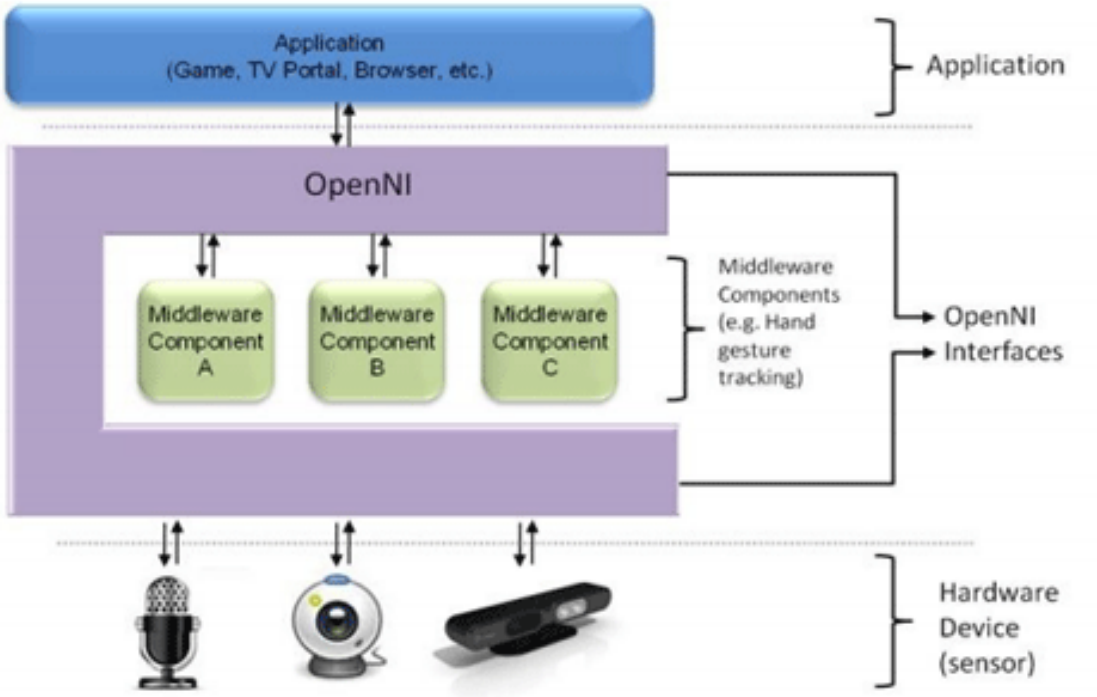
4.2 OpenNI

OpenNI SDK on avoimen lähdekoodin kehitysympäristö, joka mahdollistaa sovelluskehityksen kolmiulotteisille sensoreille. Sitä kehittää vuonna 2010 perustettu voittoa tavoittelematon OpenNI konsortio, jonka tehtävänä oman ilmoituksensa mukaan on promotoida ja standardoida luonnollisen interaktion (engl. Natural Interaction) laitteiden, ohjelmistojen ja middlewaren yhteensopivuutta. Suurin yksittäinen osallistuja OpenNI SDK:n kehitykseen on israelilainen PrimeSense -yritys, joka tunnetaan myös alkuperäisenä Kinectin tekniikan kehittäjänä. OpenNI SDK on saavuttanut vankan aseman ohjelmistonkehittäjien keskuudessa, ja sitä ladataan keskimäärin noin 85000 kertaa kuukaudessa. [30]

Tunnetuin ja käytetyin OpenNI:n tukemista kolmiulotteisista sensoreista on Microsoftin liikeohjausjärjestelmä Kinect. Sen lisäksi OpenNI tukee kuitenkin myös muita vastaavia sensoreita, kuten Asuksen Xtion, sekä PrimeSensen itsensä julkaisema Primesense sensor. [31]

OpenNI SDK tarjoaa ohjelmointirajapinnan, jonka avulla kolmiulotteisten sensorien käyttö onnistuu matalalla tasolla. Virallisena tarjotaan ainoastaan C++-kielinen rajapinta, mutta epävirallisena on tarjolla ohjelmointirajapinta ainakin Python-ohjelmointikielelle. OpenNI:n avulla voidaan kontrolloida sensoreita ja lukea niiden tuottamaa dataa. OpenNI ei itsessään tarjoa mitään esimerkiksi liikkeentunnistukseen tai konenäköön liittyviä korkeamman tason palveluita, vaan nämä on mahdollista toteuttaa käyttämällä erillisiä sitä varten kehitettyjä middleware-komponentteja. Näitä komponentteja on tarjolla sekä PrimeSensen että kolmansien osapuolten kehittäminä. SDK:n arkkitehtuuri on esitettynä kuvassa 7. [30]

OpenNI SDK on lisensoitu sallivalla Apache License 2.0 -lisenssillä, joten sitä saa vapaasti käyttää kehittämissään sovelluksissa, myös kaupallisissa, eikä omien ohjelmien lähdekoodia tarvitse julkaista. Tässä diplomityössä OpenNI SDK:ta on käytössä versio 1.5.2.23. Sitä itsessään käytetään Kinectin hallinnointiin sekä sen lähettämän kuvadatan lukemiseen. Lisäksi OpenNI tarvitaan, jotta niin ikään tarvittu NiTE-middleware saadaan käyttöön. [30]



Kuva 7: OpenNI:n arkkitehtuuri [32]

4.3 NiTE

NiTE on PrimeSensen kehittämä OpenNI SDK:n middleware-kirjasto. Se käyttää hyväkseen sensorin lähettämää syvyys- ja kuvadataa, ja tarjoaa näiden avulla perus OpenNI-toiminnallisuuteen verrattuna korkeamman tason palveluita. Sen avulla onnistuvat myös muun muassa käyttäjien erottaminen muusta kuvasta, hyvin yleisesti käytetty käyttäjän koko kehon seuranta (engl. Skeleton tracking) 14 eri kehon pisteen avulla sekä käsien ja niillä tehtävien eleiden seuranta. Käsillä tehtävistä eleistä tunnistetaan painaminen, heilutus, käden nosto sekä pyyhkäisy ylös, alas, oikealle ja vasemmalle. Eleitä on myös mahdollista määritellä lisää. [33]

NiTE tarjoaa ohjelmointirajapinnat ohjelmointikielistä C++:lle, C#:lle ja Javalle. Toisin kuin OpenNI, NiTE ei ole avointa lähdekoodia. Siitä on saatavana käännetyt binäärit Windowsille, Linuxille, Mac OS:lle ja Androidille. NiTE:n lisenssi mahdollistaa sen ilmaisen käyttämisen omissa sovelluksissa. Tässä diplomityössä NiTE:stä on käytössä versio 1.5.2. Sitä käytetään käyttäjän käden paikallistamiseen OpenNI:ltä saatavasta kuvadatasta. [33]

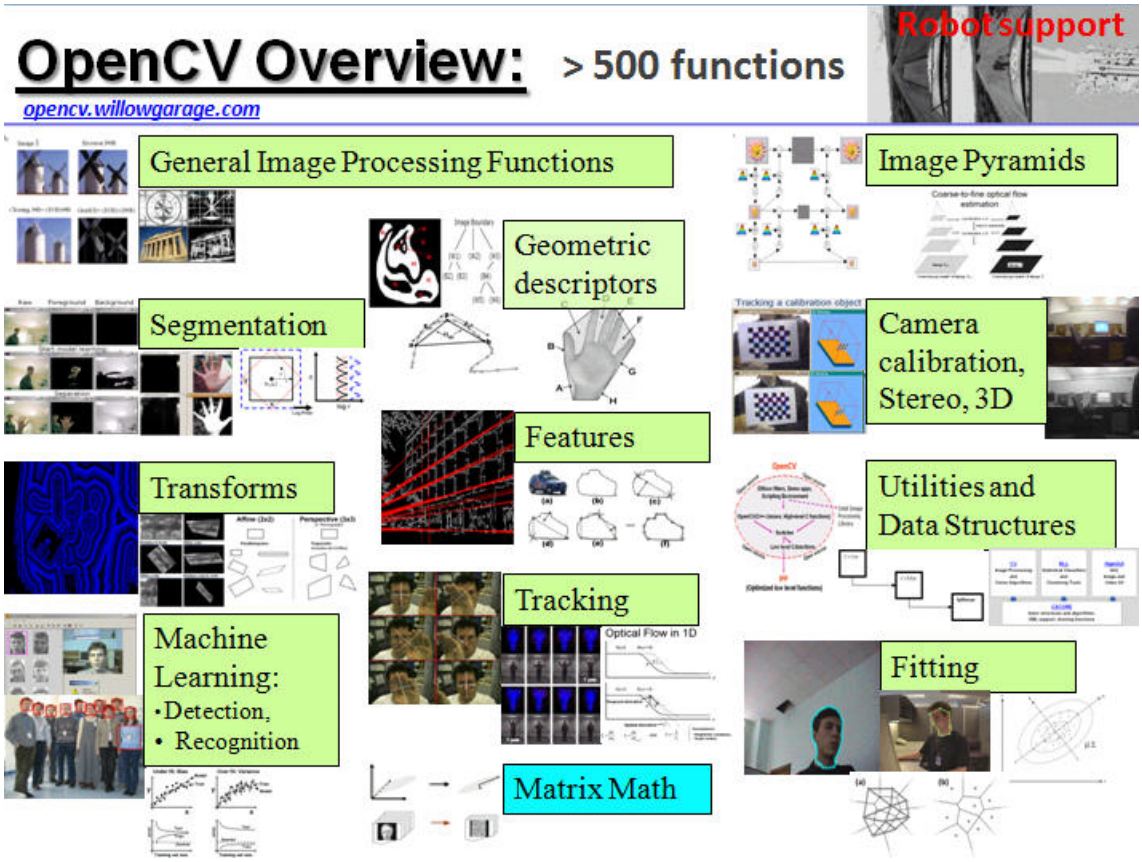
4.4 OpenCV

OpenCV on vuonna 1999 ensimmäisen kerran julkaistu avoimen lähdekoodin ohjelmistokirjasto, joka on tarkoitettu tietokonenäön toteuttamiseen. Sen on alunperin kehittänyt Intel, ja sittemmin kehitystä ja ylläpitoa ovat jatkaneet pääasiassa aktiivinen kehittäjäyhteisö ja Willow Garage tutkimusinstituutti. OpenCV on alansa tunnetuin ja suosituin ohjelmistokirjasto, ja sillä on suuret käyttäjäyhteisöt Kiinassa, Japanissa, Venäjällä, Euroopassa ja Israelissa. Kirjastoa arvioidaan ladatun yli 5 miljoonaa kertaa. Sitä käyttävät myös monet suuret yritykset, kuten Google, Yahoo, Microsoft ja IBM. Sisäisesti OpenCV on C- ja C++-pohjainen, mutta lukuisten sidosrajapintojen ansiosta sen käyttö on mahdollista kaikilla yleisimmillä ohjelmointikielillä. Käyttöjärjestelmistä tuettuja ovat Windows, Linux, Mac OS, Android ja IOS. [34]

OpenCV pyrkii helpottamaan ja nopeuttamaan tietokonenäköä käyttävien ohjelmien toteutusta. Ensisijaisesti se on suunniteltu tehokkaaksi reaaliaikaisissa ohjelmissa, joten se on toteutettu hyvin optimoidulla C- ja C++-kielisellä useammat ytimet huomioivalla ohjelmakoodilla. OpenCV sisältää yli 500 funktiota, jotka käsittävät laajan alan tietokonenäön liittyvistä osa-alueista. Koska koneoppiminen ja tietokonenäkö liittyvät läheisesti toisiinsa, OpenCV sisältää myös erillisen yleiskäyttöisen koneoppimiskirjaston (MLL). OpenCV:n eri osa-alueet on esitetty kuvassa 8. [35]

OpenCV on lisensoitu hyvin sallivalla BSD-lisenssillä, eli sen käyttö on mahdollista omissa ohjelmissa, myös kaupallisissa, eikä sitä käyttävän ohjelman tarvitse olla avointa lähdekoodia. [36]

Alunperin OpenCV on sisältänyt vain C-kielisen ohjelmointirajapinnan. Sittemmin siihen on kehitetty myös modernimpaa C++-kielistä ohjelmointirajapintaa, joka on myös geneerisempi ja helpompi käyttää kuin vanha rajapinta. Tämän diplomityön toteutuksen aikoihin C++-rajapinnassa oli kuitenkin vielä pieniä puutteita, joten toteutetussa ohjelmistokirjastossa on käytetty vanhempaa C-kielistä rajapintaa. Monia OpenCV:n tarjoamia funktioita on käytetty hyväksi, kun Kinectiltä saatavaa kuvadataa analysoimalla selvitetään, onko siinä esiintyvä käyttäjän käsi hetkellisesti tarttuneena vai ei. OpenCV:stä on käytössä versio 2.3.1.



Kuva 8: OpenCV:n osa-alueet [36]

5 TOTEUTUS

Tässä luvussa esitetään toteutetun ohjelmistokirjaston yleiskuvaus, käyttö sekä loppukäyttäjän että ohjelmistokirjastoa hyödyntävän sovelluskehittäjän näkökulmasta, ja arkkitehtuurikuvaus sekä toteutuksen sisäinen rakenne. Tämän diplomityön kannalta kaikkein oleellisin osa ohjelmistokirjastossa liittyy tarttumiseleen tunnistukseen, joten sen toiminta on kuvattu yksityiskohtaisesti omassa kohdassaan.

5.1 Yleiskuvaus

Diplomityön teknisenä kontribuutiona on toteutettu Qt-pohjainen ohjelmistokirjasto, joka mahdollistaa käyttäjän käden seurannan ja tarttumiseleen tunnistuksen Kinectillä. Kinectin hallinnointiin ja lukemiseen käytetään OpenNI-ohjelmistokirjastoa ja siihen liittyvää NiTE-middlewarea. Ohjelmistokirjasto tarjoaa suoraan NiTE:n tarjoamat käyttäjän käden seuraamisen ja painamis- sekä pyyhkäisyeleen tunnistuksen Qt-signaaleina, joilloin ne on helppoa liittää Qt-sovellukseen. Näiden lisäksi ohjelmistokirjastossa on mukana OpenCV:tä hyväksikäyttäen rakennettu tarttumiseleen tunnistus, jota käytetään diplomityöhön liittyvissä sovelluksissa. Toteutetulle ohjelmistokirjastolle on annettu nimi Air Cursor, ja sitä käytetään myös tässä tekstissä kirjastoon viitattaessa.

Air Cursor -ohjelmistokirjasto on julkaistu avoimena lähdekoodina Github-verkkopalvelussa sallivalla LGPL -lisenssillä, joka antaa sitä käyttävälle sovelluskehittäjälle vapauden käyttää sitä omissa sovelluksissaan ilman, että omaa sovellusta tarvitsisi julkaista avoimena lähdekoodina.

5.2 Käyttö

Tässä kohdassa on kuvattu Air Cursor -ohjelmistokirjaston käyttäminen loppukäyttäjän sekä Air Cursoria hyödyntävän sovelluskehittäjän näkökulmasta. Air Cursor tarjoaa käden sijaintiin ja tarttumiseleeseen liittyviä palveluita, ja sen käyttö on mahdollista monenlaisissa käyttötapauksissa. Loppukäyttäjän näkökulmasta on kuvattuna lähinnä yleisin käyttötapaus.

5.2.1 Loppukäyttäjä

Loppukäyttäjän näkökulmasta Air Cursoria käyttävän sovelluksen käyttö lähtee liikkeelle siitä, että ensin on tehtävä kädellä kohdistusele. Tämä tarvitaan, jotta NiTE-kirjasto ja sitä kautta Air Cursor pystyy seuraamaan käyttäjän kättä. Kohdistuseleenä käytössä on NiTE-kirjaston ”wave”-ele, jossa kättä täytyy liikuttaa vaakasuunnassa puolelta toiselle muutaman kerran. Kun käyttäjä tekee osittaisen kohdistuseleen, jota ei vielä voida tulki-

ta kokonaiseksi, tästä annetaan yleensä ilmoitus, jotta käyttäjä tietää jatkaa eleen suorittamista.

Kun käsi on saatu seurantaan, ilmestyy yleisessä käyttötapauksessa ruudulle kursori, esimerkiksi käden kuva, esittämään ruudulla sitä paikkaa, jolla käyttäjän seurattava käsi kulloinkin sijaitsee. Kursorin paikan lisäksi jokin, esimerkiksi kursorin muoto, kuvaa yleensä sitä, tulkitaanko käyttäjän käsi sillä hetkellä tarttuneeksi vai ei. Myös mikäli käyttäjän käsi on liian lähellä Kinectiä tai liian kaukana Kinectistä, annetaan tästä ilmoitus ruudulla. Käyttäjän käden seurannan ollessa aktiivinen käyttäjän on mahdollista ohjata sovellusta käden liikutuksella ja tarttumiseleellä.

Tarttumiseleen lisäksi tunnistetaan NiTE-kirjaston tarjoama ”push”-ele, jossa käyttäjä painaa kättään ruutua kohden, ja ”swipe”-ele, jossa käyttäjä tekee pyyhkäisyliikkeen joko vaaka- tai pystysuunnassa. Nämä eleet eivät ole tämän diplomityön kannalta oleellisia, eikä niitä ole käytetty arviointia varten toteutetuissa sovelluksissa.

Käyttäjän käden seurannan ollessa aktiivinen sitä jatketaan, kunnes käyttäjä suorittaa uudestaan ”wave”-eleen, jolloin palataan alkutilaan. Käden seuranta lopetetaan myös, jos käsi kadotetaan näkyvistä pysyvästi. Tähän tarvitaan käytännössä pidempi poistuminen Kinectin näkökentästä, eikä lyhytaikainen käden käyminen näkökentän ulkopuolella tai toisen esineen takana katkaise seurantaa.

Huomattavaa on, että Air Cursor ei tarjoa kursoria suoraan, vaan sen toteuttaminen on Air Cursoria käyttävän sovelluskehittäjän harteilla. Näin ollen onkin mahdollista, että Air Cursoria käyttävä sovellus toimii myös kokonaan ilman kursoria. Käytännössä tämä on järkevää ainoastaan sellaisissa sovelluksissa, joissa tarvittava ohjaus on hyvin suurpiirteistä, esimerkiksi vaaka- tai pystysuunnassa tapahtuvaa vieritystä. Mikäli tarvitaan hienovaraisempaa käden sijainnin paikallistamista, on kursorin käyttö lähes välttämätöntä, sillä käyttäjän on todella vaikeaa tietää, missä kohdassa hänen kätensä kulloinkin on ruudulla, ja tämä johtaa turhauttavaan käyttökokemukseen.

5.2.2 Sovelluskehittäjä

Air Cursoria omassa sovelluksessaan käyttävän sovelluskehittäjän täytyy ensin sisällyttää Qt:n projektitiedostoon tarvittavat riippuvuudet, eli OpenNi, NiTE ja OpenCV. Näiden kirjastojen lisäksi tarvitaan vielä PrimeSensen julkaisema ajuri Kinectin käyttöön, SensorKinect. Itse Air Cursor sisällytetään toteutettavaan ohjelmaan ottamalla otsikkotiedosto **aircursor.h** mukaan ohjelmakoodiin esiprosessorikäskyllä #include ja linkkaamalla mukaan ohjelmakooditiedosto **aircursor.cpp**. Air Cursorin asetuksia voidaan muuttaa suoraan ohjelmakooditiedostoa **aircursor.cpp** muokkaamalla, jossa eri asetukset on määritelty vakioarvoina tiedoston alussa. Nämä asetukset on esitetty taulukossa 2.

nimi	tyyppi	oletus-arvo	selite
GRAB_SMOOTHING_FACTOR	qreal	0.5	Tarttumiseleen tulkinnassa käytettävä tasoituskerroin.
GRAB_STATE_CHANGE_ THRESHOLD	qreal	0.1	Tarttumiseleen tulkinnassa käytettävä raja-arvo.
NUM_OF_SMOOTHING_POINTS	int	5	Kuinka montaa edellistä käden sijaintipistettä käytetään käden sijainnin pehmennyksessä.
DEPTH_MAP_SIZE_X	int	640	Syvyyskartan koko x-suunnassa.
DEPTH_MAP_SIZE_Y	int	480	Syvyyskartan koko y-suunnassa.
DEPTH_THRESHOLD	int	60	Syvyysrajauksen etäisyys. Yksikkö millimetri.
DEFECT_MIN_SIZE	int	25	Minimikoko konveksille epäkohdalle, että se lasketaan mukaan.
GRAB_MAX_DEFECTS	int	0	Maksimimäärä, joka tarttuneessa kädessä saa olla konvekseja epäkohtia
NEAR_CLIPPING_DISTANCE	int	500	Etäisyys Kinectistä, jota lähemmät pisteet syvyyskartassa hylätään. Yksikkö millimetri.
FAR_CLIPPING_DISTANCE	int	2000	Etäisyys Kinectistä, jota kauemmat pisteet syvyyskartassa hylätään. Yksikkö millimetri.
NEAR_WARNING_DISTANCE	int	700	Seuratun käden minimietäisyys Kinectistä, jota pienemmillä etäisyyksillä lähetetään varoitus-signaali. Yksikkö millimetri.
FAR_WARNING_DISTANCE	int	1700	Seuratun käden maksimietäisyys Kinectistä, jota suuremmilla etäisyyksillä lähetetään varoitus-signaali. Yksikkö millimetri.
HAND_ROI_SIZE_LEFT	int	110	Käden rajauslaatikon koko vasemmalle käden sijaintipisteestä. Yksikkö millimetri.
HAND_ROI_SIZE_RIGHT	int	110	Käden rajauslaatikon koko oikealle käden sijaintipisteestä. Yksikkö millimetri.
HAND_ROI_SIZE_UP	int	100	Käden rajauslaatikon koko ylös käden sijaintipisteestä. Yksikkö millimetri.
HAND_ROI_SIZE_DOWN	int	150	Käden rajauslaatikon koko alas käden sijaintipisteestä. Yksikkö millimetri.
CONTOUR_MIN_SIZE	int	1000	Minimikoko yhtenäiselle alueelle, jotta se lasketaan mukaan käsiehdokkaaksi.

Taulukko 2: Air Cursor -ohjelmistokirjaston asetukset.

Air Cursorin käyttö lähtee liikkeelle siitä, että **AirCursor**-luokasta instantioidaan olio. Kinectin alustukset hoidetaan **AirCursor**-olion funktiolla **init()**, jolle annetaan parametrina tieto siitä, halutaanko Air Cursorin muodostavan debug-kuvaa ajonsa aikana. Debug-kuva sisältää Kinectin tuottaman syvyyskartan, sekä käden tarttumiseleen tunnistukseen liittyviä tietoja. **Init()**-funktion paluuarvona palautetaan tieto siitä, onko alustus onnistunut. Tämän jälkeen halutut Qt-signaalit yhdistetään **AirCursor**-oliosta omaan QObjectista periytettyyn olioon. Air Cursor käynnistetään kutsumalla funktiota **run()**. Kun Air Cursor on käynnissä, voidaan sitä käyttävässä sovelluksessa siirtyä normaalisti Qt:n tapahtumasilmukkaan odottamaan tapahtumia.

Air Cursorilla on kaksi päätoimintatilaa riippuen siitä, onko käden seuranta käynnissä. Kun käden seuranta ei ole käynnissä, Air Cursor lähettää ainoastaan signaaleja **gestureProcess** ja **gestureRecognized**. **GestureProcess** lähetetään silloin, kun käyttäjän havaitaan tekevän osittainen kohdistusele. Tämä ei johda muihin toimenpiteisiin. **GestureRecognized** lähetetään silloin, kun käyttäjän havaitaan tekevän kokonainen kohdistusele. Tämän jälkeen Air Cursor siirtyy käden seurantatilaan. Ensin lähetetään signaali **handCreate**, jossa välitetään tieto käden seurannan aloittamisesta.

Käden seurannan aikana signaali **handUpdate** kertoo 30 kertaa sekunnissa seuratun käden paikan ja tiedon siitä, onko käsi tarttuneena vai ei. Kun havaitaan, että käyttäjän käsi tarttuu, lähetetään lisäksi signaali **grab**, ja kun käyttäjän käsi lopettaa tartunnan lähetetään signaali **grabRelease**. Mikäli debug-kuva on käytössä, lähetetään uusi debug-kuva ja siihen liittyvät debug-merkkijonot signaalin **debugImageUpdated** avulla 30 kertaa sekunnissa. Lisäksi kun havaitaan NiTE-kirjaston painamista vastaava ”push”- ja pyyhkäisyä vastaavas ”swipe”-ele, niistä lähetetään signaalit. Käden seurannan loppuessa ja siirryttäessä alkutilaan lähetetään signaali **handDestroy**. Kaikki Air Cursorin lähettämät signaalit on esitetty parametreineen seuraavassa listassa:

- **void handCreate(qreal x, qreal y, qreal z, qreal time)**: lähetetään, kun käyttäjän käsi havaitaan ja sen seuranta alkaa. Parametreina käden x-, y- ja z-koordinaatit reaalimaailmassa sekä havaintoaika.

- **void handDestroy(qreal time):** lähetetään, kun käyttäjän käden seuranta loppuu. Parametrina aika jolloin seuranta loppuu.

- **void gestureRecognized(QString gestureStr):** lähetetään, kun havaitaan kokonainen kohdistusele. Parametrina havaitun kohdistuseleen nimi.

- **void gestureProcess(QString gestureStr):** lähetetään, kun havaitaan osittainen kohdistusele. Parametrina osittain havaitun kohdistuseleen nimi.

- **void handUpdate(qreal x, qreal y, qreal z, qreal time, bool grab):** lähetetään, kun käyttäjän käden tila päivittyy, eli käden seurannan aikana 30 kertaa sekunnissa. Parametreina x-, y- ja z-koordinaatit, missä käsi sijaitsee, aika, ja tieto siitä, onko käsi tarttuneena vai ei.

- **void grab(qreal x, qreal y, qreal z):** lähetetään, kun käyttäjän tarttumisen tila muuttuu tilasta avoin tilaan suljettu. Parametreina käden x-, y- ja z-koordinaatit reaalimaailmassa.

- **void grabRelease(qreal x, qreal y, qreal z):** lähetetään, kun käyttäjän tarttumisen tila muuttuu tilasta suljettu tilaan avoin. Parametreina käden x-, y- ja z-koordinaatit reaalimaailmassa.

- **void handTooClose():** lähetetään, kun seurattu käsi on liian lähellä Kinect-sensoria. Säädettävissä, oletuksena varoitusetäisyys 70cm.

- **void handTooFar():** lähetetään, kun seurattu käsi on liian kaukana Kinect-sensorista. Säädettävissä, oletuksena varoitusetäisyys 170cm.

- **void debugUpdate(QImage image, QList<QString> strings):** lähetetään, kun uusi debug-kuva on saatavilla, eli debug-kuvan ollessa käytössä 30 kertaa sekunnissa. Parametreina debug-kuva sekä debug-tietoja merkkijonoina.

- **void push(qreal x, qreal y, qreal z, qreal velocity, qreal angle):** lähetetään, kun havaitaan NiTE-kirjaston painamista vastaava ”push”-ele. Parametreina eleen suorituskohdan x-, y- ja z-koordinaatit reaalimaailmassa, eleen suoritusnopeus sekä eleen suorituskulma.

- **void swipeUp(qreal velocity, qreal angle), void swipeDown(qreal velocity, qreal angle), void swipeLeft(qreal velocity, qreal angle), void swipeRight(qreal velocity, qreal angle):** lähetetään, kun havaitaan NiTE-kirjaston pyyhkäisyä vastaava ”swipe”-ele, ylös, alas, vasemmalle tai oikealle. Parametreina pyyhkäisyn nopeus ja kulma.

5.3 Arkkitehtuuri

Kirjasto koostuu kokonaisuudessaan yhdestä C++-luokasta, joka on periytetty Qt:n **QThread**-luokasta. **QThreadista** periytetyillä luokilla on monia etuja puhtaaseen C++-luokkiin verrattuna, joista etenkin kaksi on erityisen hyödyllisiä toteutetun kirjaston tapauksessa.

QThreadista periytetyllä luokalla saadaan helposti luotua uusi säie, ja tätä käytetään OpenNI- ja NiTE-osien suorittamiseen omassa säikeessään. Tämä on tärkeää, jotta Kinectin hallinta ja sen tuottaman datan lukeminen saadaan suoritettua säännöllisesti muusta ohjelmasta riippumattomasti.

Toinen tärkeä ominaisuus, jonka mahdollistaa se, että muiden valmiiden Qt:n luokkien tapaan myös **QThread** on periytetty Qt:n kantaluokasta **QObject**, on mahdollisuus käyttää Qt:n signalointijärjestelmää. Sitä käytetään tiedon välitykseen kirjaston ja sitä käyttävän ohjelman välillä, ja se soveltuu tähän tarkoitukseen erityisen hyvin kirjaston toiminnan asynkronisen luonteen vuoksi.

Kommunikointi OpenNI- ja NiTE-osista ulospäin tapahtuu callback-funktioiden avulla. Air Cursoriin on määritelty callback-funktiot, jotka rekisteröidään aluksi OpenNI / NiTE:n toiminnoille. Näin ollen aina kun OpenNI / NiTE -puolella tapahtuu jotain näihin toimintoihin liittyvää, kutsutaan vastaavia callback-funktioita Air Cursorista. Air Cursorin arkkitehtuuri on esitetty kuvassa 9.

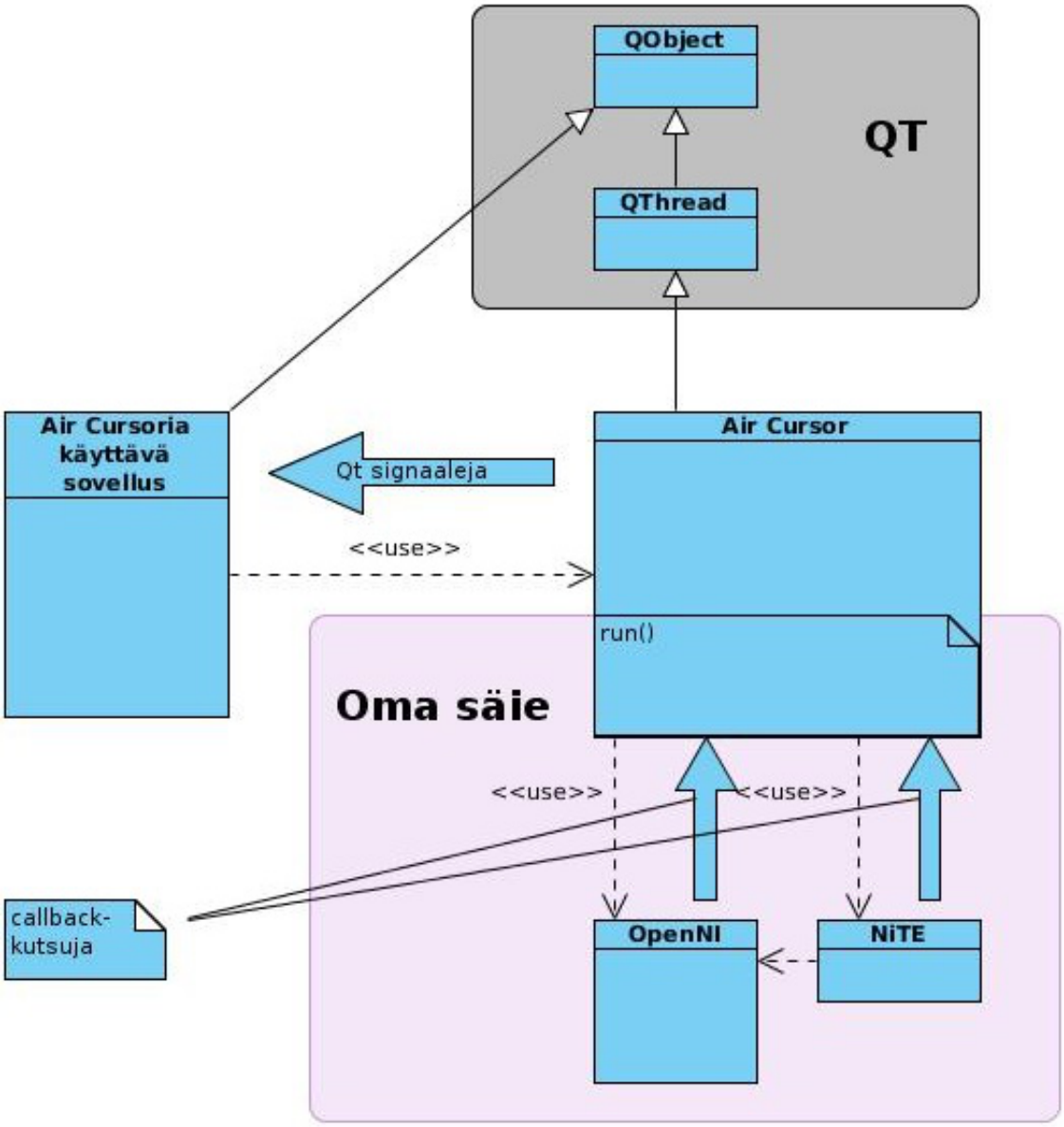
5.4 Sisäinen rakenne

Funktio **init()** hoitaa kaikki Air Cursoriin, OpenCV:hen ja Kinectiin liittyvät alustukset. Ensimmäisenä luodaan OpenNI-konteksti, joka tarvitaan kaikkeen OpenNI-toiminnallisuuteen. Kontekstiin luodaan syvyyskartan tuottamiseen tarvittava **DepthGenerator**, käsien seurantaan tarvittava **HandsGenerator** sekä eleiden tunnistamiseen tarvittava **GestureGenerator**. Tämän jälkeen luodaan OpenNI-sessio ja rekisteröidään Air Cursorin callback-funktiot OpenNI:n ja NiTE:n palveluihin sekä käynnistetään OpenNI-funktiolla **startGenerating()**.

Syvyyskarttaa varten luodaan uusi 8-bittinen OpenCV-kuva eli **IplImage**. Mikäli debug-kuva on käytössä, luodaan lisäksi myös sitä varten 24-bittinen OpenCV-kuva, jota tarvitaan sen debug-kuvan muodostamiseen ja **QImage**-kuva, jota tarvitaan debug-kuvan näyttämiseen Qt:n avulla.

Oleellinen osa Air Cursorin sisäistä toimintaa on sen sisältämä **run()**-funktio. **QThread**-luokan, josta Air Cursor-luokka periytyy, toiminta perustuu siihen, että siinä

olevan **run()**-funktion uudelleenmäärittelemällä on toteutusta mahdollista ajaa uudessa säikeessä. Air Cursorissa oleva uudelleen määritelty **run()**-funktio pitää sisällään silmukan, jota suoritetaan, kunnes Air Cursorin toiminta lopetetaan **stop()**-funktion avulla. Silmukan sisällä kutsutaan OpenNI:n syvyyskarttaa päivittävää funktiota ja sessiota päivittävää funktiota. Nämä hoitavat käytännössä OpenNI/NiTE-puolen päivityksen, ja koska ne suoritetaan run-funktiossa omassa säikeessä, ei muun ohjelman toiminta vaikuta näiden toimintaan.



Kuva 9: Air Cursor ohjelmistokirjaston arkkitehtuuri.

Kommunikointi OpenNI/NiTE-puolelta ulospäin hoituu niin, että kun niiden päivitysfunktioita kutsutaan Air Cursorin **run()**-funktiosta, kutsuvat puolestaan ne Air Cursorin callback-funktioita, mikäli erilaisia tapahtumia on tapahtunut. Callback-funktiot **handCreateCB, handDestroyCB, gestureProcessCB, gestureRecognizedCB, sessionStartCB, sessionEndCB, pushCB, swipeUpCB, swipeDownCB, swipeLeftCB, ja swipeRightCB** vastaavat suoraan OpenNI / NiTE -puolen toimintoja, ja ne ainoastaan lähettävät edelleen toimintoa vastaavan samannimisen (ilman callback-liitettä ”CB”) Qt-signaalin. Näiden signaalien merkitykset löytyvät listattuna kohdasta 5.2.2.

Kun uusi käden sijainti on saatavilla, kutsuu NiTE callback-funktiota **handUpdateCB()**. Tämä funktio toteuttaa useampia asioita. Ensimmäisenä kutsutaan Air Cursorin sisäistä funktiota **newHandPoint()**, joka lisää uuden sijaintipisteen tiedot ja laskee uuden tasoitetun sijaintipisteen. Koska käden sijaintipisteessä on värinää, on sitä käyttökokemuksen parantamiseksi edullista pehmentää. Tämä pehmennys tehdään laskeamalla useamman edellisen sijaintipisteen keskiarvo. Keskiarvon laskemiseen käytettävien sijaintipisteiden lukumäärää on mahdollista säätää Air Cursorin asetuksella **NUM_OF_SMOOTHING_POINTS**. Sen oletusarvo on 5.

Seuraavaksi **handUpdateCB()**:ssa kutsutaan Air Cursorin sisäistä funktiota **analyzeGrab()**, joka hoitaa hetkellisen tarttumiseleen analysoinnin. Kun hetkellinen tarttumistila on selvillä, kutsutaan Air Cursorin sisäistä funktiota updateState, joka määrittää uuden hetkellisen tilan perusteella tarttumiseleen tilan. Tarttumiseleen tilan selvityksen jälkeen tarkastetaan vielä, ettei käden sijainti ole liian lähellä tai liian kaukana, ja lähetetään varoitussignaali mikäli näin on.

5.5 Tarttumiseleen tunnistus

Tässä kohdassa sanalla käsi tarkoitetaan nimenomaan ranteesta sormiinpäin olevaa osaa ihmisen koko kädestä, joka sisältää kämmenen ja sormet (engl. hand). Käden asennosta käytetään nimitystä suljettu silloin, kun käsi on tarttuneena tai nyrkissä, ja nimitystä avoin muutoin.

Kinectiltä saadaan 640x480 -resoluution syvyyskartta 30 kertaa sekunnissa. Syvyyskartassa pikselin arvo on sen etäisyys millimetreissä 16-bittisenä lukuna. Tarttumiseleen tunnistus tehdään aina kun uusi syvyyskartta on saatavilla, eli 30 kertaa sekunnissa. Kuvassa 10 on kuvattuna Kinectin tuottama syvyyskartta, jossa mitä lähempänä pikseli on, sitä kirkkaammalla värillä se on esitetty.

Kuva 10: Kinectin tuottama syvyyskartta.

Seuraavissa kohdissa on tarttumiseleen tunnistuksen eri vaiheet esitetty siinä järjestyksessä, kuin ne käytännössä tapahtuvat.

5.5.1 Käden paikallistaminen syvyyskartasta

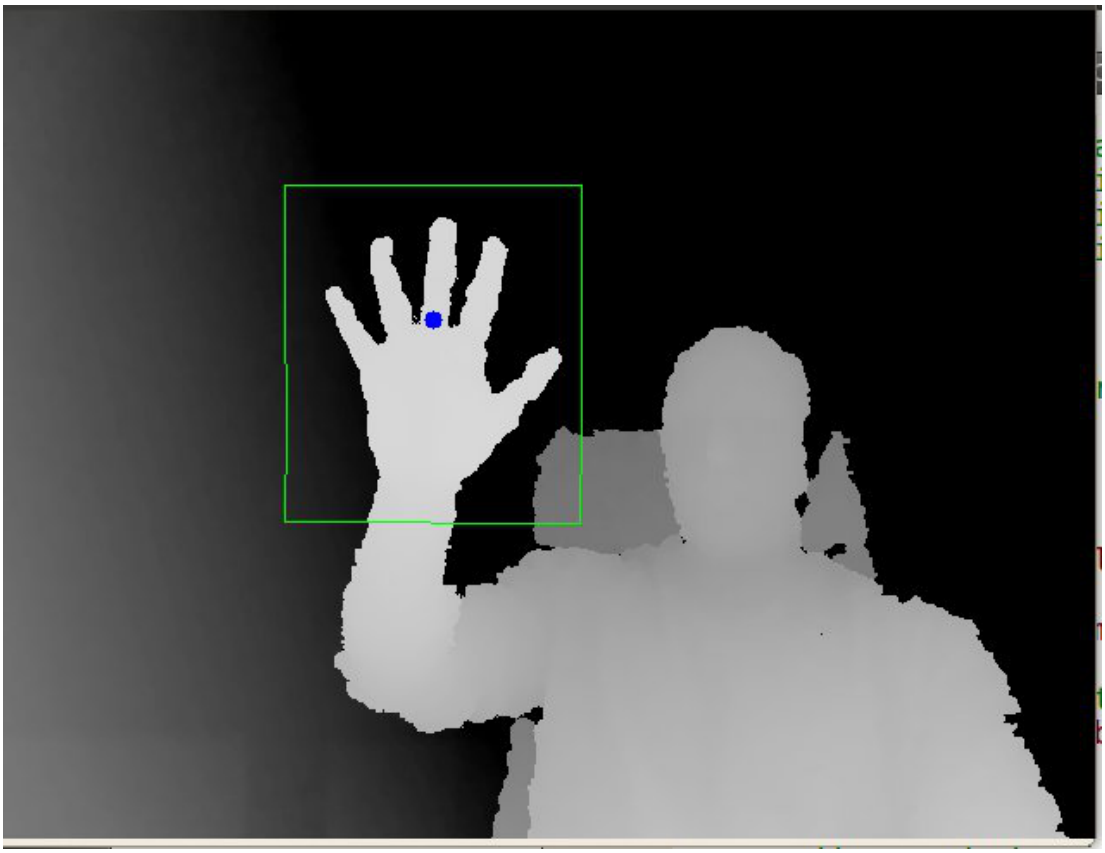
Ensimmäiseksi lähdetään liikkeelle siitä, että syvyyskartasta paikallistetaan käyttäjän käsi. Tämä tehdään NiTE-middlewaren avulla. NiTE tarjoaa palvelun, joka ilmoittaa käden sijainnin X- ja Y-koordinaatit sekä reaalimaailman koordinaatteina että pikselikoordinaatteina syvyyskartassa. NiTE ei ole avointa lähdekoodia, eikä sen toimintatekniikka ole julkista tietoa, joten ei ole mahdollista varmuudella sanoa, millä tavalla se käden paikallistamisen tekee. Käyttäjän kämmenen paikallistaminen NiTE:n avulla toimii tämän diplomityön ohessa tehtyjen testien perusteella hyvin luotettavasti.

5.5.2 Käden rajaus ja muunto siluetiksi

Seuraavaksi käsi pyritään rajaamaan mahdollisimman tarkasti erikseen ranteen kohdalta muusta raajasta, koska muuta raajaa ei tarvita otteen tunnistuksessa. Tämä tehdään kaksivaiheisesti, ensin karkeammin ja sitten tarkemmin.

Ensimmäisessä vaiheessa määritetään syvyyskartalla oleva suorakulmainen alue, jolla käsi karsitaan karkeammin. Alueena käytetään 22 senttimetriä leveää ja 26 senttimetriä korkeaa suorakulmiota, joka sijoitetaan niin, että NiTEn antama käden paikka reaalimaailman koordinaateissa on vaakasuunnassa sen keskellä, ja pystysuunnassa 10 senttimetriä sen yläreunasta ja 16 senttimetriä alareunasta. Näitä arvoja on mahdollista säätää Air Cursorin asetuksilla **HAND_ROI_SIZE_UP, HAND_ROI_SIZE_DOWN, HAND_ROI_SIZE_LEFT ja HAND_ROI_SIZE_RIGHT**. Pystysuunnassa ei käytetä keskikohtaa, koska NiTE:n antama käden paikka on pääsääntöisesti jonkun verran kämmenen keskikohtaa ylempänä.

Koska käytetyn suorakulmion koon syvyyskartassa halutaan ottavan huomioon käyttäjän käden etäisyyden, täytyy sen syvyyskarttakoordinaattien selvittämiseksi ne projektoida reaalimaailman koordinaateista pikselikoordinaatteihin. Tämä onnistuu suoraan OpenNI:n tarjoamalla funktiolla **ConvertRealWorldToProjective()**. Kuvassa 11 on esitetty NiTE:n antama käden sijainti sinisellä pisteellä, ja suorakulmainen rajausalue vihreällä.



Kuva 11: Käden sijaintipiste ja rajauslaatikko.

Toisessa vaiheessa käyttäjän käsi se pyritään rajaamaan tarkemmin käyttämällä hyväksi sitä seikkaa, että käyttäjä osoittaa ohjaavalla kädellään eteenpäin kohti Kinectiä. Tällöin etäisyys syvyyskartassa pienenee sitä mukaa, kun käyttäjän raajaa edetään olkapäästä lähtien eteenpäin. Näin käsi voidaan rajata muusta raajasta karsimalla tiettyä rajaa kauempana olevat pisteet pois. Tavoitteena olisi, että rajaus tapahtuisi mahdollisimman tarkasti käyttäjän ranteen kohdalta. Kokeellisesti hyväksi arvoksi ranteen kohdalta rajaamiseen osoittautui aikuisella noin 6 senttimetriä. Tätä arvoa voi muuttaa Air Cursorin asetuksen **DEPTH_THRESHOLD** avulla.

Seuraavaksi rajattu käsi muutetaan kaksiväriseksi eli kaksibittiseksi niin, että kaikki pisteet joilla on nollaa suurempi arvo, ja joiden siis katsotaan kuuluvan käyttäjän käteen, saavat arvon 1, ja nollat pysyvät nollina. Tämä muutos onnistuu suoraan OpenCV:n funktion **cvThreshold()** avulla. Muutos on tarpeen, koska OpenCV:n funktiot, joita jatkossa tullaan käyttämään, vaativat toimiakseen kaksivärisen kuvan. Tulos syvyysrajatusta ja kaksivärisestä kädestä on kuvattu kuvassa 12.

Kuva 12: Syvyysrajattu ja kaksibittiseksi muutettu käsi.

5.5.3 Kättä vastaava alue, konveksi peite ja konveksit epäkohdat

Seuraavaksi otteen tunnistus etenee niin, että kuvasta määritetään yhtenäiset alueet OpenCV:n funktiolla **cvFindContours()**, joka palauttaa yhtenäisten alueiden ääriviivat. Näistä alueista suurimman voidaan olettaa vastaavan käyttäjän kättä, joten alueiden koot määritetään funktiolla **cvContourSize()**, ja suurin valitaan jatkokäsittelyyn. Hyvin pienet alueet, joiden voidaan olettaa olevan häiriöitä eivätkä näin ollen varmasti vastaava käyttäjän kättä, voidaan karsia suoraan pois säätämällä Air Cursorin asetusta **CONTOUR_MIN_SIZE**, jonka oletusarvo on 1000.

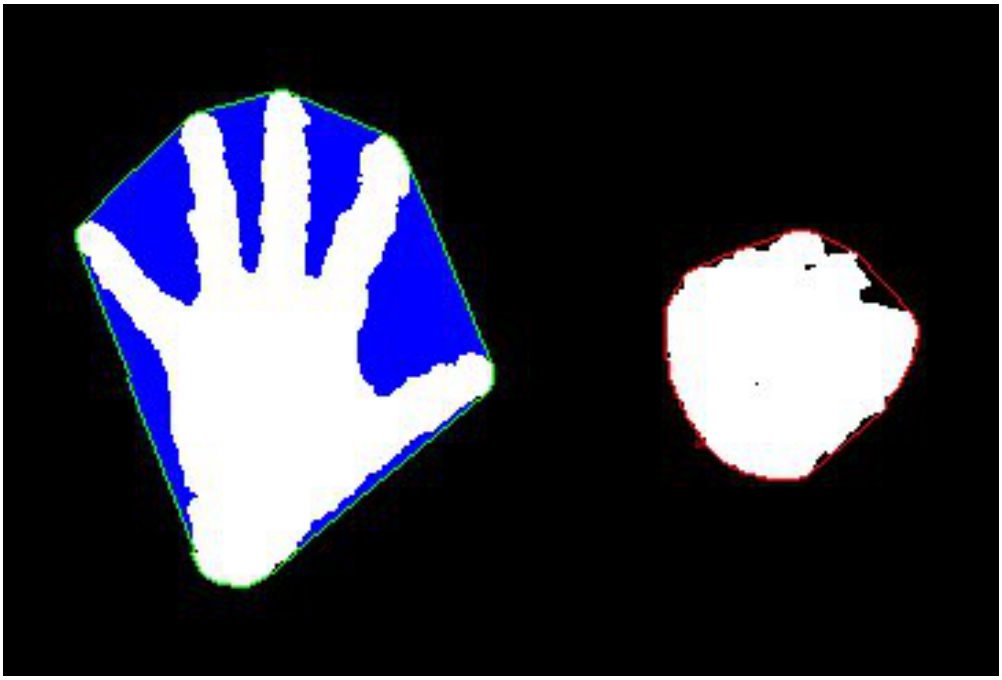
Kun kättä vastaava alue on selvillä, määritellään sen avulla käden kaksiulotteinen konveksi peite. Konveksi peite tarkoittaa pienintä mahdollista konveksia aluetta, joka sisältää kaikki halutut pisteet, eli tässä tapauksessa käyttäjän kättä vastaavan alueen ääriviivojen pisteet. Konveksi alue taas määritellään niin, että kaikkien siihen kuuluvien pisteiden välille piirretyn janan täytyy kokonaisuudessaan pysyä alueen sisällä.

Sormet pyritään seuraavaksi erottamaan kädestä selvittämällä käyttäjän käden konveksin peitteen konveksit epäkohdat (engl. Convexity defects), eli eroja joita esiintyy, kun kättä kuvaavaa pistejoukkoa verrataan sen konveksiin peitteeseen. Tätä menetelmää käyttämällä voidaan analysoida monia monimutkaisia kuvioita. Konveksit epäkohdat esiintyvät kuvassa ikäänkuin rakoina, joten käden tapauksessa niitä voidaan käyttää avoinaisen käden sormien välien paikallistamiseen.

Konveksit epäkohtien selvittämiseen käytetään OpenCV:n funktiota **cvConvexityDefects()**. Koska käden kuva ei ole täydellisen konveksi peite, täytyy pienimmät konveksit epäkohdat rajata pois, jotta jäljelle jäävät epäkohdat tosiaan osoittavat sormien välejä. Konveksin epäkohdan kokoon voi määrittää palautuvalla depth-arvolla, joka kertoo etäisyyden konveksin peitteen reunan ja siitä kaukaisimman epäkohdan pisteen välillä. Minimikoon konvekseille epäkohdille, jotka tulkitaan esittämään käyttäjän sormien välejä, voi määrittää Air Cursorin asetuksella **MIN_DEFECT_SIZE**. Kokeellisesti hyväksi oletusarvoksi on osoittautunut 25.

Edelleen kun sormien välien lukumäärä tiedetään, voidaan tästä tiedosta helposti päätellä esillä olevien sormien lukumäärä lisäämällä sormien välien lukumäärään yksi. Tarttumisotteen tunnistus tehdään suoraan tämän sormien lukumäärän perusteella. Mikäli konvekseja epäkohtia ei ole yhtään, sormet ovat kiinni toisissaan ja käden tulkitaan olevan tarttumisasennossa, muutoin käden tulkitaan olevan avoinna. Kiinniolevassa kädessä sallittujen konveksien epäkohtien määrää voi säätää Air Cursorin asetuksella **GRAB_MAX_DEFECTS**, mutta käytännössä juuri 0 osottautui selvästi toimivimmaksi. Kuvassa 13 vasemmalla on kuvattuna avoimen käden konveksi peite rajattuna vih-

reällä, ja oikealla suljetun käden konveksi peite rajattuna punaisella. Lisäksi kuvassa on merkitty avoimen käden konveksit epäkohdat sinisellä.



Kuva 13: Avoimen ja suljetun käden konveksi peite sekä avoimen käden konveksit epäkohdat

5.5.4 Tasoitus useamman hetkittäisen tilan välillä

Syvyyskartassa ja siten myös siitä muodostettavassa käden kuvassa esiintyy jatkuvasti pieniä häiriöitä, jotka aiheuttavat ajoittaisia virheellisiä tulkintoja sormien määrässä. Tarttumisohjauksessa tällainen virheellinen tulkinta voi yksittäisenäkin olla monesti käytettävyyden kannalta hyvin haitallinen. Esimerkiksi tilanteessa, jossa raahataan elementtiä, yksikin virheellinen käden avoimeksi tulkitseminen päästää elementin irti, jolloin se saattaa päätyä väärään paikkaan, tai vähintäänkin se usein palaa automaattisesti lähtöpaikkaansa, joka on käyttäjälle hyvin ärsyttävää.

Näiden häiriöiden vaikutusta voidaan vähentää, kun lopullista päätöstä käden asennosta ei tehdä pelkästään sen hetkisen yksittäisen tilan perusteella, vaan päätöksessä otetaan myös edeltäviä tiloja, jolloin yksittäiset häiriöt tasoittuvat pois. Tähän käytetään eksponentiaalista tasoitusta. Se toimii niin, että jatkuvasti pidetään kirjaa käden asennon sen hetkisestä numeerisesta tasoitetusta arvosta. Tasoitettu arvo lasketaan kaavalla

$$s_0 = x_0$$
$$s_t = \alpha * x_t + (1 - \alpha) * s_{t-1}$$

jossa **t** on ajanhetki, **s** tasoitettu arvo, **α** tasoituskerroin ja **x** hetkellinen käden asento. Käytännössä tasoitettu arvo muodostetaan siis niin, että ensimmäisen näytteen tapauk-

sessa tasoitettu arvo on suoraan ensimmäisen näytteen arvo, ja siitä eteenpäin tasoitettu arvo on uuden näytteen ja edellisen tasoitetun arvon painotettu keskiarvo. **x**:n arvo määritetään siten, että se on 1 käden ollessa kiinni, ja 0 käden ollessa auki. **α**:n arvo valitaan väliltä [0..1]. Mitä pienempi se on, sitä vähemmän yksittäiset uudet arvot vaikuttavat tasoitettuun arvoon. Toimivaksi **α**:n arvoksi määritettiin kokeellisten testien perusteella 0.5. **α**:n arvoa voi säätää Air Cursorin asetuksella **GRAB_SMOOTHING_FACTOR**.

Kaavasta nähdään, että koska **x** ja **α** ovat välillä [0..1], myös tasoitettu arvo pysyy aina sillä välillä. Tasoitetun arvon mukaan tehdään lopullinen päätös, onko käsi kiinni vai auki. Air Cursorin kehityksen alkuvaiheessa valinta tehtiin suoraan yhden raja-arvon perusteella niin, että sitä suuremmilla arvoilla käden tilaksi tuli kiinni ja pienemmillä auki. Tässä tuli kuitenkin esille, että käden asennossa esiintyi paljon virheellistä värinää tasoitetun arvon ollessa raja-arvon läheisyydessä, koska pienetkin muutokset tasoitetussa arvossa saattoivat muuttaa käden asennon.

Paremmaksi ratkaisuksi osoittautui kahden erillisen raja-arvon menetelmä niin, että kun edellinen asento on kiinni, tasoitetun arvon täytyy alittaa tietty arvo jotta asento vaihdetaan avonaiseksi, ja vastaavasti edellisen asennon ollessa auki, täytyy tasoitetun arvon ylittää tietty arvo jotta asennoksi vaihdetaan kiinni. Käytännössä siis käden asennon pitää olla selkeä, jotta se aiheuttaa tilan muutoksen. Tarvittavaa poikkeamaa puolivälistä 0.5 voi säätää Air Cursorin asetuksella **GRAB_STATE_CHANGE_THRESHOLD**. Kokeellisten testien perusteella sille määritettiin toimiva alkuarvo 0.1.

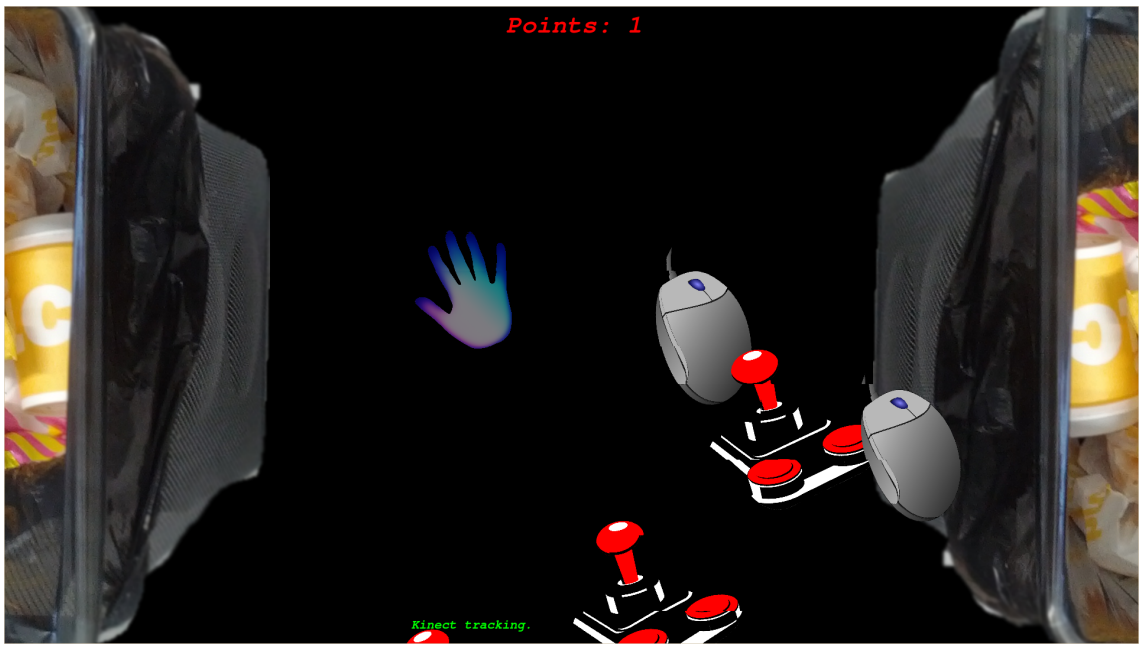
6 ARVIOINTI

Tässä luvussa arvioidaan kehitetyn ohjelmistokirjaston toimivuutta ja tarttumisohjauksen yleistä käytettävyyttä, sekä esitetään jatkokehitysajatukset.

6.1 Arvioinnissa käytetyt sovellukset

Air Cursor -ohjelmistokirjaston toimintaa ja tarttumiseleeseen perustuvan liikeohjauksen käytettävyyttä arvioitiin kahden ohjelmistokirjastoa käyttävän sovelluksen avulla. Toinen oli varta vasten ohjelmistokirjaston testaukseen kehitetty yksinkertainen kaksiulotteinen peli ja toinen elektroninen kanban-seinä, jota ohjattiin liikeohjauksella. Molemmissa näistä liikeohjaus toimi samankaltaisesti. Käyttäjän tuli ensin suorittaa kohdistusele, jotta käden seuraaminen alkoi. Käden seurauksen ollessa käynnissä sen paikka ruudulla näytettiin kättä muistuttavalla kursorikuvalla. Kursorin ulkonäkö muuttui lisäksi sen mukaan, onko käsi tarttuneena vai ei.

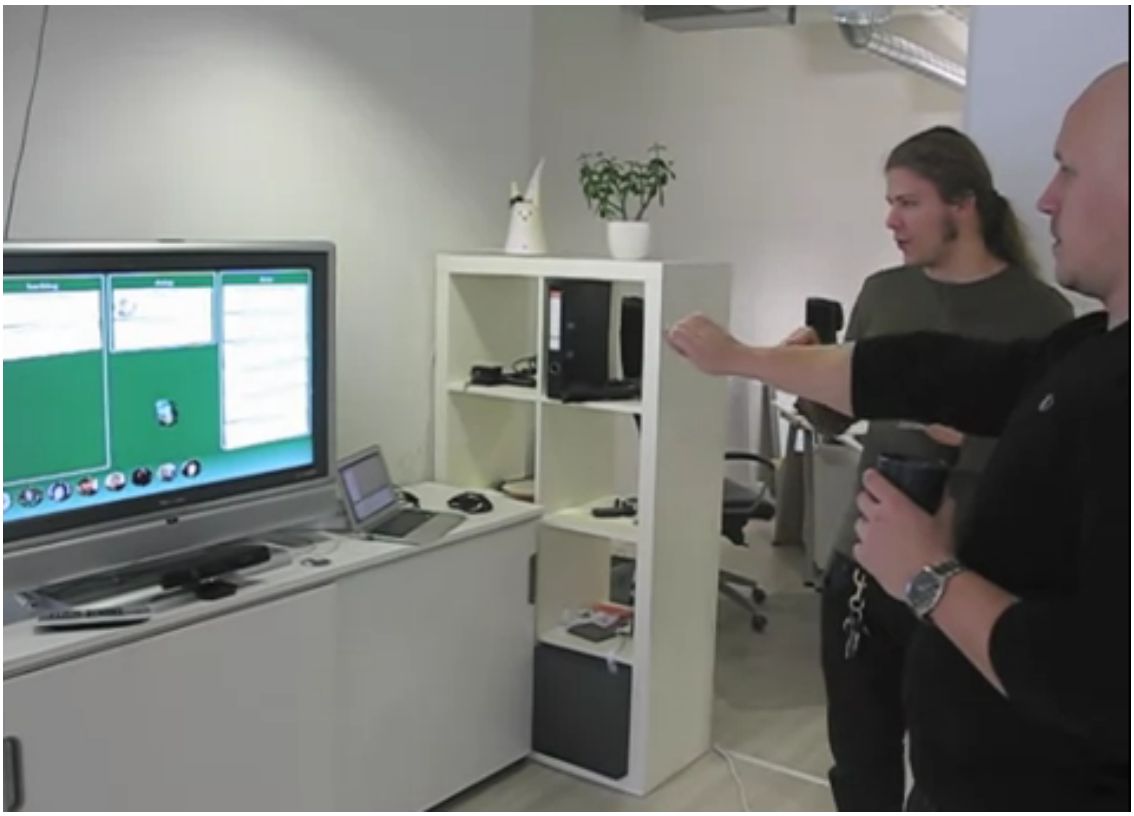
Kaksiulotteisessa pelissä oli tarkoituksena tarttua ruudulla liikkuviin objekteihin ja siirtää ne ruudun vasemmassa tai oikeassa reunassa sijaitsevaan lokeroon ennen kuin ne pääsevät etenemään ruudun alalaidasta ylälaitaan. Objektien liikkumis- ja ilmestymisnopeus nousivat pelin edetessä. Peli suunniteltiin nopeatempoiseksi, jolloin sillä saatiin selvitettyä, kuinka hyvin tarttumisohjauksen käyttö onnistui uusilla käyttäjillä paineen alla. Peliä testattiin noin viidellä kymmenellä käyttäjällä Qt Contributor -tapahtumassa Berliinissä kesäkuussa vuonna 2012. Kuvassa 14 on ruutukaappaus pelistä.



Kuva 14: Ruutukaappaus arvioinnissa käytetystä Grab to the future -pelistä

Perinteinen kanban-seinä koostuu taululle piirretyistä lokeroista sekä Post-it -lapuista, joita siirrellään näiden lokeroiden välillä. Elektronisessa versiossa tämä tehdään graafisesti siirtelemällä objekteja lokerosta toiseen. Testijärjestelyissä näyttönä käytettiin 32-tuuman LCD-televisiota. Kanban-sovellus oli noin kymmenen hengen kokeiltavana muutaman kuukauden ajan pienessä suomalaisessa it-alan yrityksessä. Kuvassa 15 on kuvattuna kanban-sovelluksen käyttötilanne.

Arvioinnit toteutetun ohjelmistokirjaston toimivuudesta ja tarttumisohjauksen yleisestä käytettävyydestä on tehty seuraamalla testihenkilöiden toimintaa testisovellusten kanssa. Osaa testihenkilöistä on myös haastateltu vapaamuotoisesti. Tehdyt havainnot ja haastatteluissa esiin nousseet yksityiskohdat ovat olleet pääsääntöisesti hyvin samanlinjaisia, joten saatuja tuloksia voidaan pitää uskottavina, vaikka testihenkilöitä ei etenkään kanban-sovelluksen kanssa ollutkaan suurta määrää.



Kuva 15: Arvioinnissa käytetty kanban-sovellus käytössä

6.2 Tarttumiseleen tunnistus

Pääsääntöisesti tarttumiseleen tunnistus toimi hyvin. Etenkin sen jälkeen, kun käyttäjiä oli nopeasti muutamalla lauseella opastettu, miten kättä kannattaa pitää ja missä kohdalla kannattaa seistä, testisovelluksien ohjaaminen onnistui vähintäänkin tyydyttävästi. Käyttötilannetestit osoittivat nopeasti, että Air Cursor -ohjelmistokirjaston toteuttama liikeohjaus ei ole vielä toistaiseksi tarpeeksi virheetöntä tuotantokäyttöön. Kuitenkin koska suuri osa siihen liittyvistä ongelmista saatiin ratkaistua muutamalla pikaisella neuvolla käyttäjille, se soveltui hyvin tarttumiseleeseen perustuvan liikeohjauksen yleiseen arviointiin.

Kello ranteessa tai pitkät hihat aiheuttivat sen, että ranteen paikan määrittäminen tuotti ajoittain virheellistä tietoa, jolloin käden erottelu kuvasta ja sitä kautta tarttumisotteen tunnistaminen toimivat virheellisesti. Nämä eivät kuitenkaan rikkoneet tunnistusta kokonaan, ainoastaan vaikeuttivat sitä hetkellisesti. Lisäksi ne oli helppo korjata testitarkoituksissa käärimällä hihat tai käyttämällä kättä, jossa ei ole kelloa.

Tarttumiseleen tunnistus perustuu sormien välien laskemiseen, joten jos käyttäjä pitää kätensä levynä ja sormet täysin yhdessä, käsi tulkitaan virheellisesti olemaan tarttuneena, koska sormien välejä ei pystytä havaitsemaan. Tämä ei kuitenkaan ennakko-oletusten vastaisesti osoittautunut käytännössä ongelmaksi. Testeissä huomattiin, että testihenkilöt pitivät oletusarvoisesti sormiaan sen verran erillään toisistaan, ettei virheellisiä tulkintoja siihen liittyen esiintynyt.

Yksi suurimmista tarttumisohjauksen käyttöön liittyneistä rajoitteista oli sen käyttöalue. Parhaan toiminnan saavuttamiseksi etäisyys käyttäjästä Kinectiin täytyy olla välillä 0.7m – 1.7m. Lisäksi on erittäin tärkeää, että käyttäjä seisoo keskellä suorassa linjassa Kinectin kohdalla, ja osoittaa kädellään kohti. Esimerkiksi ohjaus käyttöalueen sivulta tai istuen pöydän ääressä toimii huonosti.

Myöskin luonnollisin tarttumisote, jossa kaikki sormien päät puristetaan yhteen toisiinsa, tuottaa ajoittain ongelmia. Tämä johtuu siitä, että tällöin Kinectin tuottamassa syvyyskartassa ei ole käden kohdalla tasaisia pintoja, vaan irrallisia pisteitä. Koska tarttumiseleen tunnistuksessa käden oletetaan olevan tasainen pinta, aiheuttaa tämä jonkun verran virhetulkintoja. Virhetulkintojen vaikutusta saadaan pienennettyä käyttämällä kohdassa 5.5.4 kuvattua pehmennystä useamman hetkittäisen tilan välillä, jolloin yksittäiset virhetulkinnat eivät vielä vaihda käden tilaa. Tarttuminen laittamalla käden nyrkkiin toimii paremmin, mutta se taas ei ole niin luonnollinen käyttää.

6.3 Tarttumisohjauksen käyttö

Tarttumisohjauksen käytännöllisyyden saama palaute testihenkilöiden keskuudessa oli yleisesti positiivista. Monien mielestä myös kosketusnäytön korvaaminen tarttumiseleeseen perustuvalla liikeohjauksella tuntui mahdolliselta vaihtoehdolta, varsinkin sen jälkeen, kun tarttumiseleen tunnistus saadaan hiottua toimimaan aukottomasti. Tarttumista pidettiin myös kätevämpänä ja luonnollisempana tapana valita objekteja ruudulta kuin Kinectin valmiina tarjoamia tapoja, kuten kursorin pitämistä paikallaan objektin päällä.

Näiden palautteiden kanssa täytyy kuitenkin ottaa huomioon, että suurin osa testihenkilöistä käytti vastaavanlaista ohjaimetonta liikeohjausta ensimmäistä kertaa, jolloin alkuinnostus saattaa selittää osan positiivisesta palautteesta. Elektronisen kanban-seinän käyttäjien keskuudessa, jotka käyttivät sovellusta pidemmän aikaa, palaute ei ollut yhtä positiviista kuin testipelin, mutta sekin oli kuitenkin selvästi pääsääntöisesti positiivista.

Testipelin käyttäjistä 76% onnistui käyttämään sovellusta itsenäisesti ilman ohjeistusta, jonka perusteella tarttumisohjausta voidaan pitää intuitiivisena tapana ohjata sovellusta. Lisäksi 20% käyttäjistä saavutti pelissä ensimmäisellä pelikerrallaan pistemäärän, jota voidaan pitää hyvänä suorituksena. Tämä osoittaa, että myös tarttumisohjauksen vaativampi käyttö onnistuu ilman harjoitusta suhteellisen hyvin.

Tarttumisohjauksen liittyvistä havaituista ongelmista suurimpana nousi esiin kosketusnäyttöjen yhteydessäkin tuttu niin sanottu gorilla arm syndrome. Tällä viitataan tilanteeseen, jossa käsi väsyy kosketusnäyttöä käyttäessä, kun se joutuu olemaan pitkiä aikoja jännitettynä vaakatasossa, eikä pääse lepäämään välillä. Sitä pidetään yhtenä suurimpana syynä, miksi kosketusnäytölliset sovellukset eivät ole tehneet läpimurtoa kotitietokoneilla. [37]

Gorilla arm syndrome esiintyy kyseessä olevan kaltaisen liikeohjauksen kanssa mahdollisesti vielä pahempana kuin kosketusnäytöllä, koska kättä ei liikeohjattua sovellusta käytettäessä saa tuettua edes kosketusnäyttöön. Monilla testihenkilöillä jo muutaman minuutin yhtämittainen käyttö aiheutti väsymistä ja kipeytymistä käsissä. Kun liikeohjausta käyttää enemmän, alkavat kädet tottua siihen, eikä väsyminen ole enää niin paha ongelma kuin alussa. On kuitenkin on selvää, ettei käden liikutteluun perustuva liikeohjaus sovellu käytettäväksi sovelluksissa, jotka vaativat pidempiaikaista jatkuvaa ohjausta.

Tarttumisohjauksen käytön luonnollisuuteen liittyen osa testihenkilöistä koki ongelmaksi myös vasteen puutteen tehdyissä liikkeissä ja eleissä. Normaalisti tartuttaessa esineeseen sen muoto tunnetaan kädessä, kun taas ohjaimettomassa tarttumisohjauksessa tarttuminen täytyy suorittaa olemattomaan esineeseen. Kosketusnäytöllä tilanne on pa-

rempi, koska siinä näyttöä täytyy fyysisesti koskettaa, joka aiheuttaa vasteen käyttäjän painalluksille. Vasteen puuttuminen vaikutti kuitenkin olevan ainoastaan ensimmäisten käyttökertojen ongelma johon tottuu nopeasti, eivätkä testisovelluksia pidempään käyttäneet enää pitäneet sitä suurena ongelmana.

6.4 Jatkokehitys

Tarttumisohjaukseen kehitetyn ohjelmistokirjaston tarkoituksena oli paitsi selvittää, onnistuuko tarttumiseleen tunnistus Kinectillä, selvittää myös miten käytettävä tällainen ohjaus on. Näin ollen kun tarttumiseleen tunnistus saatiin kehitettyä riittävän hyvälle tasolle, ei aikaa enää käytetty sen viilaamiseksi paremmaksi, vaan keskityttiin itse ohjaustavan testaukseen. Tarttumiseleen tunnistusmekanismiin olisi vielä mahdollista kehittää monia parannuksia.

Yksinkertaisimpiin parannuksiin lukeutuisi värierottelun käyttäminen apuna kämmenen erottamisessa muusta kädestä. Käyttäjän kämmen rajattaisiin siis syvyysrajauksen lisäksi myös ihonväriin perustuen. Tällä saataisiin parannusta etenkin ongelmia aiheuttaneisiin tilanteisiin, joissa käyttäjällä on päällään pitkähihainen löysä paita tai kädessään rannekello.

Analysoinnissa, kuvaako hetkellinen käden kuva tarttunutta vai avointa kättä, voitaisiin käyttää apuna oppivia järjestelmiä, esimerkiksi neuroverkkoja, sen sijaan, että käden tarttuminen päätettäisiin suoraan kuvan konveksien epäkohtien perusteella. Tällöin käden kuvasta valittaisiin tiettyjä kättä hyvin kuvaavia ominaisuuksia, joilla ensin opetettaisiin järjestelmää, minkälaiset arvot näillä ominaisuuksilla vastaavat tarttunutta ja avonaista kättä, ja sen jälkeen järjestelmä voisi itse tunnistaa arvojen perusteella, onko käsi tarttuneena. Oppivia järjestelmiä on käytetty menestyksekkäästi apuna vastaavanlaisissa kuvan luokitteluun liittyvissä tehtävissä. Ongelman muodostavat käytettävien ominaisuuksien valitseminen ja neuroverkon rakenteen määrittely. Lisäksi opettamiseen käytettävää opetusdataa tarvitaan paljon. [38]

Käytetyssä tarttumisen analysointialgoritmissa on paranneltunakin se ongelma, että se perustuu kaksiulotteisen kuvan analysointiin, jolloin kuvasta katoaa kolmannen ulottuvuuden informaatio. Tällaisesta kaksiulotteisesta kuvasta on hyvin vaikeaa tunnistaa tarttumista tietyissä ongelmakohdissa, kuten esimerkiksi jo mainitussa tilanteessa, jossa sormet ovat yhdessä, mutta silti suorana. Näin ollen paras ratkaisu algoritmin parantamiseksi olisi vaihtaa se kokonaan kolmiulotteiseen malliin pohjautuvaksi, jossa Kinectiltä saatavasta kolmiulotteisesta datasta luotaisiin suoraan kättä ja sormia kuvaavat kolmiulotteiset objektit. Kun tällainen malli saadaan toimimaan luotettavasti, on tarttumisen tarkastelu triviaalia vertaamalla sormenpäiden etäisyyttä toisiinsa. Kolmiulotteisten

mallien käyttöön liittyy kuitenkin myös paljon ongelmia, kuten teknisen toteutuksen haastavuus, ja tarvittavan laskentatehon suuri määrä.

Valitaan tarttumiseleen tunnistukseen sitten Air Cursorin tapainen yksinkertaisempi kuvan analysointi, tai monimutkaisempi kolmiulotteisiin malleihin perustuva tapa, on ilmeistä, että suurimmat ongelmat tunnistuksessa aiheuttaa Kinectin syvyyskartan pieni resoluutio. Tämä myös rajoittaa käyttöetäisyyden pieneksi. Näin ollen mahdollinen jatkokehitys kannattaa Kinectin sijaan suunnata tulevalle Kinect 2 -laitteelle, jossa voidaan ennakkotietojen perusteella olettaa olevan tarkempiresoluutioinen syvyyskuvaus, joka helpottaa tarttumiseleen tunnistusta merkittävästi. [39]

7 YHTEENVETO

Tässä diplomityössä tutkittiin, miten tarttumiseleen tunnistus onnistuu Microsoft Kinectillä vaikkei laite sitä itse tuekaan, ja olisiko tarttumiseleeseen perustuvasta liikeohjauksesta mahdollisesti kosketusnäytöllä tapahtuvan drag-and-drop -ohjauksen korvaajaksi.

Teknisenä kontribuutiona toteutettiin avoimen lähdekoodin Qt-pohjainen ohjelmistokirjasto, joka OpenCV-tietokonenäkökirjastoa hyväksikäyttäen mahdollisti tarttumisohjauksen käytön Qt-sovelluksissa. Tarttumisohjauksen testaukseen toteutettiin lisäksi kaksi tätä kirjastoa käyttävää sovellusta, yksinkertainen kaksiulotteinen peli sekä elektroninen versio projektinhallintaan käytettävästä kanban-seinästä. Peli oli nopeatempoinen ja sitä testaamassa noin viisikymmentä ihmistä, elektroninen kanban-seinä taas hidastempoinen, ja sitä testasi kymmenkunta ihmistä pidemmällä kuukausia kestäneellä jaksolla.

Tuloksena tarttumiseleen tunnistukseen liittyen saatiin, että pääsääntöisesti se toimi tyydyttävästi. Virhetulkintoja tarttumisotteen tilassa esiintyi jonkin verran, eikä toimintavarmuus näin riitä vielä mihinkään kaupalliseen käyttöön. Suurimmaksi osaksi objektien siirtely kuitenkin onnistui, ja toimintavarmuutta saatiin myös parannettua huomattavasti ottamalla rajoitukset huomioon. Toimintavarmuus oli näin ollen täysin riittävänä itse ohjaustavan arviointiin.

Suurimpia ongelmia toiminnassa aiheutti järjestelmän rajoittunut käyttöalue, joka edellytti, että käyttäjä oli 0.7 metrin – 1.7 metrin päässä Kinectistä ja pysyi sivusuunnassa keskellä. Muita ongelmia aiheuttivat muutamat erikoistapaukset, kuten käyttäjän paidassa olevat pitkät löysät hihat. Myös erilaisilla tarttumisotteilla oli vaikutusta tarttumiseleen tunnistuksen toimimiseen, eikä yleisesti ihmisille luonnollisempi tapa tarttua painamalla sormien päät yhteen toiminut niin hyvin, kuin vähemmän luonnollinen käden laittaminen nyrkkiin.

Tarttumiseleeseen perustuvaa liikeohjausta pidettiin hauskana tapana ohjata sovellusta. Suurin osa ensikertalaisista testihenkilöistä onnistui sen käytössä ilman suurempaa ohjeistusta, joten sitä voidaan myös pitää intuitiivisena. Testit toivat esille myös erittäin suuren ongelman, joka liittyy kaikkeen kädellä tehtävään liikeohjaukseen; kädet väsyvät siinä todella nopeasti. Vaikka tämä väheneekin kun käyttöön tottuu, ei tämänlainen liikeohjaus sovellu käytettäväksi pidempiä aikoja kerrallaan ilman lepoa.

Tarttumiseleen tunnistukseen olisi mahdollista tehdä monia parannuksia. Yksinkertaisilla parannuksilla, kuten värierottelun lisäämisellä käyttäjän käden karsintaan, saataisiin nykyiseen tunnistukseen parannuksia helposti. Tulevaisuudessa laskentatehon ja

sensorien resoluution parantuessa menetelmä kannattaisi korvata kokonaan kolmiulotteisiin malleihin perustuvalla tunnistusmekanismilla. Kun sormien liikkeitä saadaan seurattua kolmiulotteisesti, on tarttumiseleen tunnistus triviaalia.

Alkuperäiseen kysymykseen siitä, onko tarttumiseleeseen perustuvasta liikeohjauksesta kosketusnäyttöjen korvaajaksi kaksiulotteisissa objektien siirtelyyn perustuvissa sovelluksissa voidaan todeta, ettei tälle ole mitään estettä, olettaen että tulevaisuudessa itse eleen tunnistus saadaan hiottua virheettömälle tasolle. Tarttumisohjauksella onnistuvat samat asiat kuin kosketusnäytöllä, mutta sitä eivät kuitenkaan koske samat näytön kokoon ja sijoitteluun liittyvät rajoitukset.

LÄHTEET

[1] Rouse, M. 2011. Motion Gaming Definition. Online. http://whatis.techtarget.com/definition/motion-gaming-motion-controlled-gaming Viitattu 21.1.2013.

[2] Oh, B. 2012. Motion sensors de-mystified. Online. http://www.edn.com/design/sensors/4402401/Motion-sensors-de-mystified Viitattu 22.1.2013.

[3] Macphee K., Hunt H. 1999. Online. Galloping gyroscopes. http://plus.maths.org/content/galloping-gyroscopes Viitattu 11.2.2013.

[4] Bernstein, J. 2003. An Overview of MEMS Inertial Sensing Technology. Online. http://www.sensorsmag.com/sensors/acceleration-vibration/an-overview-mems-inertial-sensing-technology-970 Viitattu 11.2.2013.

[5] Renaudin, V., Afzal M. & Lachapelle, G. New Method for Magnetometers Based Orientation Estimation, IEEE/ION PLANS, Palm Springs, California, USA, MAY 4-6, 2010.

[6] Weng, K. W. 2009. Measuring Tilt Angle with Gyro and Accelerometer. Online. http://tutorial.cytron.com.my/2012/01/10/measuring-tilt-angle-with-gyro-and-accelerometer/ Viitattu 11.2.2013.

[7] Erol, A., Bebis, G., Nicolescu, M, Boyle, R. & Twombly, X. Vision-based hand pose estimation: A review. Computer Vision and Image Understanding 108 (2007), pp. 52-73.

[8] Pavlovic, V., Sharma, R. & Huang, T. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. IEEE transactions on pattern analysis and machine intelligence 19 (1997) 7, pp. 681.

[9] Oxford, N. 2010. The History of Motion Controls. Online. http://gametheoryonline.com/2010/12/06/motion-controls-game-gaming-history/ Viitattu 7.3.2013.

[10] Mann, P. 2009. Motion-sensing game controllers explained. Online. http://hexus.net/tech/tech-explained/peripherals/19458-motion-sensing-game-controllers-explained/ Viitattu 22.1.2013.

[11] PlayStation Move Motion Controller delivers a whole new entertainment experience to PlayStation 3. Online. http://scei.co.jp/corporate/release/100311e.html Viitattu 12.2.2013.

[12] Shelah, S. 2010. Microsoft names Israel's PrimeSense as Project Natal partner

http://www.globes.co.il/serveen/globes/docview.asp?did=1000550536&fid=942 Viitattu 12.5.2013.

[13] Pc pro. 2012. Microsoft Kinect for Windows. Online. http://www.pcpro.co.uk/features/378043/microsoft-kinect-for-windows Viitattu 12.5.2013.

[14] The Verge. Microsoft Kinect. Online. http://www.theverge.com/products/kinect/1792. Viitattu 12.5.2013.

[15] Kinect Sales Surpass Ten Million 2001. Online. http://www.xbox.com/en-US/Press/archive/2011/0308-Ten-Million-Kinects. Viitattu 22.1.2013.

[16] Kinect body tracking troubleshoot. Online. https://support.xbox.com/en-US/xbox-360/kinect/body-tracking-troubleshoot Viitattu 13.2.2013.

[17] Microsoft. 2010. Kinect manual. pp. 4-5.

[18] Everybody plays. 2010. How to set up Kinect. Online. http://www.everybodyplays.co.uk/feature/360/How-to-set-up-Kinect/402 Viitattu 13.5.2013.

[19] Kinect for Windows Sensor Components and Specifications. http://msdn.microsoft.com/en-us/library/jj131033.aspx. Viitattu 24.1.2013.

[20] LeadBetter, R. 2010. PrimeSense: Beyond Natal. Online. http://www.eurogamer.net/articles/digitalfoundry-primesense-article Viitattu 19.2.2013.

[21] MirrorImage. 2010. How Kinect depth sensor works – stereo triangulation? On line. http://mirror2image.wordpress.com/2010/11/30/how-kinect-works-stereo-triangulation/ Viitattu 28.1.2013.

[22] Wikipedia Kinect. Online. http://en.wikipedia.org/wiki/Kinect/ Viitattu 28.3.2013.

[23] Adafruit blog. WE HAVE A WINNER – Open Kinect driver(s) released. 2011. Online. http://www.adafruit.com/blog/2010/11/10/we-have-a-winner-open-kinect-drivers-released-winner-will-use-3k-for-more-hacking-plus-an-additional-2k-goes-to-the-eff/. Viitattu 24.1.2013.

[24] Catone, J. 2010. Microsoft Now "Excited" by Kinect Hacks. Online. http://mashable.com/2010/11/20/microsoft-kinect-hacks/. Viitattu 27.1.2013.

[25] Terdiman, D. 2011. Kinect developer claims credit for hack bounty idea. Online. http://news.cnet.com/8301-13772_3-20034579-52.html. Viitattu 27.1.2013.

[26] Blanchette, J. & Summerfield, M. 2008. C++ GUI Programming with Qt 4. Prentice Hall, Westford MA, USA. ISBN 0-13-235416-0 pp preface xv-xvi, 20-22, 589-590. p 718.

[27] Qt overview. Online. http://www.developer.nokia.com/Community/Wiki/Qt_overview Viitattu 28.3.2013.

[28] Paul, R. Nokia releases Qt 4.7 with terrific new mobile UI framework. Online.

http://arstechnica.com/information-technology/2010/09/nokias-cross-platform-development-strategy-evolves-with-qt-47/ Viitattu 3.5.2013.

[29] Qt Licensing. Online. http://qt.digia.com/licensing Viitattu 28.3.2013.

[30] About OpenNI. Online. http://www.openni.org/about/ Viitattu 14.2.2013.

[31] OpenNI. 3D Sensors. Online. http://www.openni.org/3d-sensors/ Viitattu 25.4.2013.

[32] Loriot, Y. 2011. Kinect: How to install and use OpenNI on Windows – Part 1. Online. http://yannickloriot.com/2011/03/kinect-how-to-install-and-use-openni-on-windows-part-1/ Viitattu 28.3.2013.

[33] NiTE TM Middleware − the Natural Interaction Engine . 2012. Primesense Ltd. pp 2-4.

[34] About OpenCV. Online. http://opencv.org/about.html Viitattu 27.2.2013.

[35] Bradski, G. & Kaehler, A. 2008. Learning OpenCV. O'Reilly Media, Sebastopol, CA, USA. ISBN 978-0-596-51613-0, pp 1-2. p 555.

[36] OpenCV Wiki. Online. http://opencv.willowgarage.com/wiki/ Viitattu 28.3.2013.

[37] Pogue, D. 2013. Why Touch Screens Will Not Take Over. Online. https://www.scientificamerican.com/article.cfm?id=why-touch-screens-will-not-take-over Viitattu 10.5.2013

[38] Guoqiang, P. Neural Networks for Classification: A Survey. IEEE transactions on systems, man, and cybernetics – part c: applications and reviews 30 (2000) 4, pp. 454.

[39] Vgleaks. 2013. Durango Next-Generation Kinect Sensor. Online. http://www.vgleaks.com/durango-next-generation-kinect-sensor/ Viitattu 13.5.2013.